# Final Project 1 Report

Group: **B**

Participants: **Atera Alam & Priska Mohunsingh**

Project Title: **Optimizing Patient Care: A Hospital Readmissions Database**

### Milestone 1:

Dataset:

The first dataset is collected from kaggle and the second dataset is collected from centers for medicare and medicaid services.
**Dataset 1 Link:** https://www.kaggle.com/datasets/dubradave/hospital-readmissions
**Dataset 2 Link:** https://data.cms.gov/provider-data/dataset/9n3s-kdb3

Problem Definition:

In this project, we will build an operational database to collect and store some patient data when they are admitted into the hospital. The goal for this database is to store data that will help in determining the trend and likelihood of patient readmissions. Patient readmission in hospitals can occur due to a suboptimal treatment plan and lack of quality patient care. Collecting this data will further help in predictive analysis of the likelihood of patient readmission. Furthermore, it will provide insight on various other helpful analytical viewpoints, such as common factors for a successful treatment.

Objectives:

- Reduce hospital readmission rates by using predictive analytics to identify patients at high risk of unplanned readmissions.
- Improve operational efficiency by tracking KPIs like length of stay, bed availability, and resource allocation.
- Enhance patient outcomes by identifying key factors leading to successful treatments and lower readmission rates.
- Support decision-making for healthcare administrators by providing insights into treatment effectiveness, staffing needs, and cost management.

Business Context:

Hospital readmissions are a significant challenge in the healthcare industry, often leading to increased operational costs and penalties under healthcare regulations. Developing an internal operational database focused on hospital readmissions will help healthcare institutions manage patient data more efficiently, while also generating valuable insights for improving care delivery.

This project aims to:

- **Streamline data management**: Efficient data storage, modification, and retrieval.

# Final Project 1 Report

- **Ensure HIPAA compliance**: The database will be designed with data privacy and security in mind, protecting sensitive patient information.
- **Provide real-time insights**: Operational teams will have access to data-driven insights to improve patient flow and treatment planning.

### Technical Approach:

We aim to build the operational database using a relational database management system (DBMS), such as PostgreSQL. The database will be designed to handle both structured and semi-structured data, focusing on scalability and data security. The following are key technical components:

- Database Schema Design: A well-structured relational schema will be designed to store patient demographics, medical history, treatment details, and readmission data.
- ETL Pipeline: An ETL process will be established to collect and clean the data before it's loaded into the database.
- Predictive Analytics: Machine learning models, such as logistic regression and decision trees, will be applied to predict readmission likelihood based on patient data.
- Data Privacy & Security: HIPAA-compliant protocols will be implemented to ensure sensitive data is encrypted and securely stored.

### Challenges and Solutions:

Some challenges include the possibility of inconsistent or incomplete data that can hinder analysis. We will implement data cleaning techniques to ensure the accuracy and reliability of the stored data.

As the database grows, ensuring fast query performance is essential. Thus, we will design the database with normalization in mind and use indexing for high-performance queries.

### Project Analysis Overview:

We aim to analyze several different relationships within the data. Some examples include being able to create KPIs like tracking the percentage of unplanned readmission rates and treatment completions, length of stay optimization, bed availability, cost analysis, and resources available for staffing.

### Data Collection:

Some of the key data points that will be collected and analyzed in the database include:

Patient Demographics: Age group, gender

# Final Project 1 Report

Medical History: Number of outpatient, inpatient, and emergency room visits prior to hospitalization
Treatment Information: Labs requested, procedures performed, medications administered
Physician Details: Specialty, procedures performed
Readmission Data: Whether the patient was readmitted, and the factors leading to readmission
KPIs for Analysis: Readmission rates, length of stay, cost analysis, and staffing resources

| age | time_in_hospital | n_lab_procedures | n_procedures | n_medications | n_outpatient | n_inpatient | n_emergency | medical_specialty | diag_1 | diag_2 | diag_3 | glucose_test | A1Ctest | change | diabetes_med | readmitted |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

- “age" - age bracket of the patient
- "time_in_hospital" - days (from 1 to 14)
- "n_procedures" - number of procedures performed during the hospital stay
- "n_lab_procedures" - number of laboratory procedures performed during the hospital stay
- "n_medications" - number of medications administered during the hospital stay
- "n_outpatient" - number of outpatient visits in the year before a hospital stay
- "n_inpatient" - number of inpatient visits in the year before the hospital stay
- "n_emergency" - number of visits to the emergency room in the year before the hospital stay
- "medical_specialty" - the specialty of the admitting physician
- "diag_1" - primary diagnosis (Circulatory, Respiratory, Digestive, etc.)
- "diag_2" - secondary diagnosis
- "diag_3" - additional secondary diagnosis
- "glucose_test" - whether the glucose serum came out as high (> 200), normal, or not performed
- "A1Ctest" - whether the A1C level of the patient came out as high (> 7%), normal, or not performed
- "change" - whether there was a change in the diabetes medication ('yes' or 'no')
- "diabetes_med" - whether a diabetes medication was prescribed ('yes' or 'no')
- "readmitted" - if the patient was readmitted at the hospital ('yes' or 'no')

Static and Transactional Data:

**Static Reference Data:** Data that provides context to the transactional data and doesn’t change as frequently. Below are the static data that we will be incorporating in our project.

# Final Project 1 Report

- **Patient Information:** Patient ID, Demographics, Contact Information and Insurance information.
- **Healthcare Providers:** Provider ID, Provider, Name, Field of Specialty, Contact Information.
- **Hospital Departments:** Department ID, Name, Location within the hospital, Services.
- **Medication Information:** Medication ID, Name, Dosage Forms, Manufacturer

**Transactional Data:** Transactional data is dynamic data that changes from more frequently than static data. Either a transaction or an event triggers the updating of this data.

The list below contains the transaction data that our project will use.

**Transactional Data** (dynamic data capturing events or transactions):

- **Admission Attributes:**
    i. Admission ID
    ii. Patient ID (Foreign Key)
    iii. Admission Date and Time
    iv. Reason for Admission
    v. Admitting Department ID (Foreign Key)
    vi. Attending Provider ID (Foreign Key)
- **Discharge Attributes:**
    vii. Discharge ID
    viii. Admission ID (Foreign Key)
    ix. Discharge Date and Time
    x. Discharge Disposition (e.g., Home, Transferred)
    xi. Follow-up Instructions
- **Medical Procedures Attributes:**
    xii. Procedure Record ID
    xiii. Admission ID (Foreign Key)
    xiv. Procedure Code (Foreign Key)
    xv. Procedure Date and Time
    xvi. Performing Provider ID (Foreign Key)
    xvii. Procedure Outcome
- **Medications Administered Attributes:**
    xviii. Medication Administration ID
    xix. Admission ID (Foreign Key)
    xx. Medication ID (Foreign Key)
    xxi. Dosage
    xxii. Route of Administration
    xxiii. Administration Date and Time
- **Laboratory Test Attributes:**
    xxiv. Lab Test ID
    xxv. Admission ID (Foreign Key)

# Final Project 1 Report

xxvi. Test Type
xxvii. Test Code
xxviii. Test Date and Time
xxix. Results

- **Readmission Records Attributes:**
  xxx. Readmission ID
  xxxi. Original Admission ID (Foreign Key)
  xxxii. Readmission Date
  xxxiii. Readmission Reason
  xxxiv. Associated Costs
- **Billing Records Attributes:**
  xxxv. Billing ID
  xxxvi. Admission ID (Foreign Key)
  xxxvii. Total Charges
  xxxviii. Insurance Provider ID (Foreign Key)
  xxxix. Claim Status
  xl. Patient Out-of-Pocket Expenses
- **Patient Feedback Attributes:**
  xli. Feedback ID
  xlii. Admission ID (Foreign Key)
  xliii. Satisfaction Score
  xliv. Comments
- **Staff Scheduling Attributes:**
  xlv. Schedule ID
  xlvi. Provider ID (Foreign Key)
  xlvii. Shift Date
  xlviii. Shift Start and End Times
  xlix. Assigned Department ID (Foreign Key)
- **Equipment Usage Logs Attributes:**
  l. Usage Log ID
  li. Equipment ID (Foreign Key)
  lii. Admission ID (Foreign Key)
  liii. Usage Start and End Times
  liv. Maintenance Issues Reported

Stakeholder Impact:

This project will benefit multiple stakeholders. By providing real-time data on patient flow and treatment effectiveness, **administrators** can make more informed decisions on staffing, resource allocation, and patient care strategies. The system will provide insights into which treatments and procedures lead to fewer readmissions, helping **providers** improve the quality of care. A more efficient healthcare system ensures **patients** receive timely, effective care, reducing the risk of readmission and improving overall patient outcomes.

# Final Project 1 Report

### Milestone 2

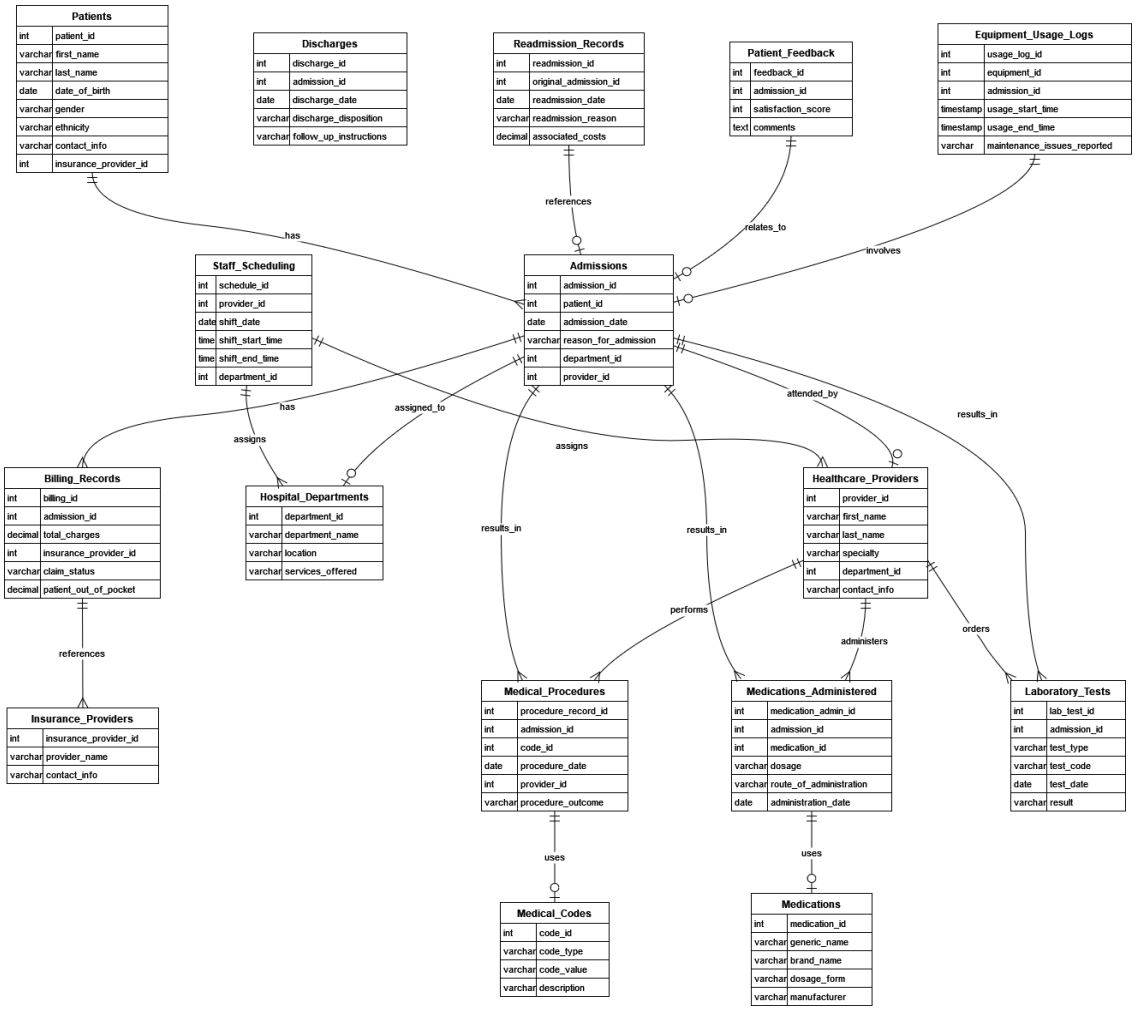
### Business Context

Hospital readmissions present a significant challenge in healthcare, contributing to higher operational costs and potential penalties due to regulatory non-compliance. Our project aims to optimize patient care by developing an operational database that collects, stores, and analyzes patient readmission data. This database will assist healthcare administrators and clinicians in identifying key factors that lead to readmissions, thus improving patient outcomes, reducing costs, and enhancing operational efficiency.

The primary goal is to use this data for predictive analytics, identifying patients who are at a higher risk of unplanned readmissions and understanding the operational drivers behind successful treatments and reduced readmission rates.

# Final Project 1 Report

## Extended Entity-Relationship Diagram (ERD)




### Static Reference Data Entities

- Patients
- Healthcare Providers
- Hospital Departments
- Medications
- Insurance Providers
- Medical Codes

### Transactional Data Entities

- Admissions
- Discharges
- Medical Procedures
- Medications Administered
- Laboratory Tests
- Readmission Records
- Billing Records

# Final Project 1 Report

- Patient Feedback
- Staff Scheduling
- Equipment Usage Logs

**Entity Relationships**

- Patients can have multiple Admissions.
- Admissions are associated with one Patient, one Healthcare Provider, and one Hospital Department.
- Admissions can lead to multiple Medical Procedures, Medications Administered, and Laboratory Tests.
- Healthcare Providers perform Medical Procedures, administer Medications, and order Laboratory Tests.
- Medications Administered reference Medications.
- Billing Records are linked to Admissions and Insurance Providers.
- Readmission Records are associated with previous Admissions.
- Patient Feedback is linked to Admissions.
- Staff Scheduling connects Healthcare Providers to Hospital Departments.
- Equipment Usage Logs link Equipment to Admissions.

### Mapping ERD to Relational Model and Normalization

After creating the ERD, we mapped it to a relational model by defining tables for each entity and specifying relationships through primary and foreign keys.

**Relational Model Tables**

1. Patients
   - patient_id (Primary Key)
   - first_name
   - last_name
   - date_of_birth
   - gender
   - ethnicity
   - contact_info
   - insurance_provider_id (Foreign Key)
2. Healthcare Providers
   - provider_id (Primary Key)
   - first_name
   - last_name
   - specialty
   - department_id (Foreign Key)
   - contact_info
3. Hospital Departments
   - department_id (Primary Key)

# Final Project 1 Report

- department_name
- location
- services_offered

4. Medications
    - medication_id (Primary Key)
    - generic_name
    - brand_name
    - dosage_form
    - manufacturer
5. Insurance Providers
    - insurance_provider_id (Primary Key)
    - provider_name
    - contact_info
6. Medical Codes
    - code_id (Primary Key)
    - code_type (Diagnosis, Procedure)
    - code_value
    - description
7. Admissions
    - admission_id (Primary Key)
    - patient_id (Foreign Key)
    - admission_date
    - reason_for_admission
    - department_id (Foreign Key)
    - provider_id (Foreign Key)
8. Discharges
    - discharge_id (Primary Key)
    - admission_id (Foreign Key)
    - discharge_date
    - discharge_disposition
    - follow_up_instructions
9. Medical Procedures
    - procedure_record_id (Primary Key)
    - admission_id (Foreign Key)
    - code_id (Foreign Key)
    - procedure_date
    - provider_id (Foreign Key)
    - procedure_outcome
10. Medications Administered
    - medication_admin_id (Primary Key)
    - admission_id (Foreign Key)
    - medication_id (Foreign Key)
    - dosage

# Final Project 1 Report

- route_of_administration
- administration_date

11. Laboratory Tests
    - lab_test_id (Primary Key)
    - admission_id (Foreign Key)
    - test_type
    - test_code
    - test_date
    - result
12. Readmission Records
    - readmission_id (Primary Key)
    - original_admission_id (Foreign Key)
    - readmission_date
    - readmission_reason
    - associated_costs
13. Billing Records
    - billing_id (Primary Key)
    - admission_id (Foreign Key)
    - total_charges
    - insurance_provider_id (Foreign Key)
    - claim_status
    - patient_out_of_pocket
14. Patient Feedback
    - feedback_id (Primary Key)
    - admission_id (Foreign Key)
    - satisfaction_score
    - comments
15. Staff Scheduling
    - schedule_id (Primary Key)
    - provider_id (Foreign Key)
    - shift_date
    - shift_start_time
    - shift_end_time
    - department_id (Foreign Key)
16. Equipment Usage Logs
    - usage_log_id (Primary Key)
    - equipment_id (Foreign Key)
    - admission_id (Foreign Key)
    - usage_start_time
    - usage_end_time
    - maintenance_issues_reported

**Normalization**

# Final Project 1 Report

Tables then needed to be normalized to reduce redundancy:

- First Normal Form (1NF): All tables have primary keys, and all columns contain atomic values.
- Second Normal Form (2NF): All non-key attributes are fully functionally dependent on the primary key.
- Third Normal Form (3NF): No transitive dependencies; non-key attributes depend only on the primary key.

### Implementing Relational Model in PostgreSQL

Two table creation examples below:

```
CREATE TABLE Patients (
       patient_id SERIAL PRIMARY KEY,
       first_name VARCHAR(50),
       last_name VARCHAR(50),
       date_of_birth DATE,
       gender VARCHAR(10),
       ethnicity VARCHAR(50),
       contact_info TEXT,
       insurance_provider_id INTEGER REFERENCES
InsuranceProviders(insurance_provider_id)
);
CREATE TABLE Admissions (
       admission_id SERIAL PRIMARY KEY,
       patient_id INTEGER REFERENCES Patients(patient_id),
       admission_date DATE,
       reason_for_admission VARCHAR(255),
       department_id INTEGER REFERENCES HospitalDepartments(department_id),
       provider_id INTEGER REFERENCES HealthcareProviders(provider_id)
);
```

### Identifying Potential Dimensions, Hierarchies, and Measures

**Dimensions**

- Patient Dimension
    a. Attributes: patient_id, age_group, gender, ethnicity, insurance_provider_id
    b. Hierarchy: Age Group > Gender > Ethnicity
- Time Dimension
    a. Attributes: date, day, month, quarter, year
    b. Hierarchy: Day > Month > Quarter > Year
- Provider Dimension
    a. Attributes: provider_id, specialty, department_id
    b. Hierarchy: Provider > Specialty > Department

# Final Project 1 Report

- Diagnosis Dimension
    a. Attributes: code_id, code_value, description
    b. Hierarchy: Code Value > Code Type (Diagnosis/Procedure)
- Geographic Dimension
    a. If you have location data for patients or providers.
    b. Hierarchy: City > State > Region

**Measures**

- Number of Readmissions
- Average Length of Stay
- Total Charges
- Patient Satisfaction Scores
- Medication Usage Counts

**Hierarchies**

- Organizational Hierarchy: Department > Division > Hospital
- Time Hierarchy: Hour > Day > Week > Month > Year

**Slowly Changing Dimensions (SCDs)**

- Patient Insurance Information: Could change over time (Type 2 SCD).
- Provider Roles or Departments: Providers may switch departments (Type 2 SCD).

```
import psycopg2
import pandas as pd
from faker import Faker
# Database connection parameters
db_params = {
   'host': 'localhost',
   'database': 'Project 1 - Healthcare Database',
   'user': 'user',
   'password': 'password'
}

# Connect to PostgreSQL database
conn = psycopg2.connect(**db_params)
cursor = conn.cursor()
```

# Final Project 1 Report

```
# Initialize Faker for synthetic data generation
fake = Faker()

# Generate synthetic insurance providers
insurance_providers = []
for _ in range(10):
    provider = (
        fake.company(),
        fake.address()
    )
    insurance_providers.append(provider)

# Convert to DataFrame
insurance_df = pd.DataFrame(insurance_providers, columns=['provider_name',
'contact_info'])

# Insert data into InsuranceProviders table
for index, row in insurance_df.iterrows():
    cursor.execute(
        """
        INSERT INTO InsuranceProviders (provider_name, contact_info)
        VALUES (%s, %s)
        """,
        (row['provider_name'], row['contact_info'])
    )
conn.commit()
print("InsuranceProviders table populated successfully.")

departments = [
    ('Emergency', 'First Floor', 'Emergency care services'),
    ('Cardiology', 'Second Floor', 'Heart-related treatments'),
    ('Neurology', 'Third Floor', 'Brain and nervous system services'),
    ('Orthopedics', 'Second Floor', 'Bone and muscle treatments'),
    ('Oncology', 'Fourth Floor', 'Cancer treatments'),
    ('Pediatrics', 'First Floor', 'Child healthcare services')
]

# Define new departments
new_departments = [
    ('InternalMedicine', 'Third Floor', 'Internal medicine treatments'),
```

# Final Project 1 Report

```
    ('Family/GeneralPractice', 'First Floor', 'General practice services'),
    ('Surgery', 'Second Floor', 'Surgical procedures and operations'),
    ('Emergency/Trauma', 'Ground Floor', 'Emergency and trauma care'),
    ('General Medicine', 'Second Floor', 'General medical services')
]

departments_df = pd.DataFrame(departments, columns=['department_name', 'location',
'services_offered'])

# Fetch department IDs from the database
cursor.execute("SELECT department_id, department_name FROM HospitalDepartments")
departments = cursor.fetchall()
department_mapping = {dept_name: dept_id for dept_id, dept_name in departments}

# Generate synthetic healthcare providers
providers = []
specialties = ['Emergency Medicine', 'Cardiology', 'Neurology', 'Orthopedics',
'Oncology', 'Pediatrics']
for _ in range(20):
    specialty = fake.random_choices(elements=specialties, length=1)[0]
    department_id = department_mapping.get(specialty.split()[0], None)
    provider = (
        fake.first_name(),
        fake.last_name(),
        specialty,
        department_id,
        fake.phone_number()
    )
    providers.append(provider)

providers_df = pd.DataFrame(providers, columns=['first_name', 'last_name',
'specialty', 'department_id', 'contact_info'])

# Insert data into HealthcareProviders table
for index, row in providers_df.iterrows():
    cursor.execute(
        """
        INSERT INTO HealthcareProviders (first_name, last_name, specialty,
department_id, contact_info)
        VALUES (%s, %s, %s, %s, %s)
```

# Final Project 1 Report

```
        """,
        (row['first_name'], row['last_name'], row['specialty'], row['department_id'],
row['contact_info'])
    )

conn.commit()
print("HealthcareProviders table populated successfully.")

medications = [
    ('Atorvastatin', 'Lipitor', 'Tablet', 'Pfizer'),
    ('Levothyroxine', 'Synthroid', 'Tablet', 'AbbVie'),
    ('Lisinopril', 'Prinivil', 'Tablet', 'Merck'),
    ('Metformin', 'Glucophage', 'Tablet', 'Bristol-Myers Squibb'),
    ('Amlodipine', 'Norvasc', 'Tablet', 'Pfizer'),
    ('Metoprolol', 'Lopressor', 'Tablet', 'Novartis')
]

medications_df = pd.DataFrame(medications, columns=['generic_name', 'brand_name',
'dosage_form', 'manufacturer'])


# Insert data into Medications table
for index, row in medications_df.iterrows():
    cursor.execute(
        """
        INSERT INTO Medications (generic_name, brand_name, dosage_form, manufacturer)
        VALUES (%s, %s, %s, %s)
        """,
        (row['generic_name'], row['brand_name'], row['dosage_form'],
row['manufacturer'])
    )

conn.commit()
print("Medications table populated successfully.")

# Medical Code Generation and Integration

medical_codes = [
    ('Diagnosis', 'E11.9', 'Type 2 diabetes mellitus without complications'),
```

# Final Project 1 Report

```
    ('Diagnosis', 'I10', 'Essential (primary) hypertension'),
    ('Procedure', '0W3P0ZZ', 'Insertion of Infusion Device into Subcutaneous Tissue'),
    ('Procedure', '3E0A3MZ', 'Introduction of Other Antineoplastic into Peripheral
Vein'),
]

medical_codes_df = pd.DataFrame(medical_codes, columns=['code_type', 'code_value',
'description'])

# Insert data into MedicalCodes table
for index, row in medical_codes_df.iterrows():
    cursor.execute(
        """
        INSERT INTO MedicalCodes (code_type, code_value, description)
        VALUES (%s, %s, %s)
        """,
        (row['code_type'], row['code_value'], row['description'])
    )

conn.commit()
print("MedicalCodes table populated successfully.")

## Static Patient Table
# Load Kaggle dataset
kaggle_df = pd.read_csv("/Users/ateraalam/Desktop/DATA ANALYTICS ENGINEERING/IE
6750/PROJECT_1_Datasets/hospital_readmissions.csv")

# Generate unique patient IDs
file_path = "/Users/ateraalam/Desktop/DATA ANALYTICS ENGINEERING/IE
6750/PROJECT_1_Datasets/hospital_readmissions.csv"

kaggle_df['patient_id'] = kaggle_df.index + 1

# Estimate date_of_birth based on age bracket
import pandas as pd
from datetime import datetime

def estimate_dob(age_range):
    # Clean up unwanted characters from the age range
```

# Final Project 1 Report

```
    age_range = age_range.replace('[', '').replace(']', '').replace(')',
'').replace('(', '').strip()
    age_range = age_range.split('-')

    if len(age_range) == 2:
        try:
            # Calculate the average age
            age = int((int(age_range[0].strip()) + int(age_range[1].strip())) / 2)
            # Estimate the year of birth
            birth_year = pd.Timestamp.now().year - age
            # Create a date object, assuming January 1st as the birth date
            date_of_birth = datetime(birth_year, 1, 1).date()
            return date_of_birth
        except ValueError:
            print(f"Invalid age range: {age_range}")
            return None
    else:
        print(f"Unexpected format in age range: {age_range}")
        return None
kaggle_df['date_of_birth'] = kaggle_df['age'].apply(estimate_dob)

# Generate synthetic gender and contact_info
kaggle_df['gender'] = kaggle_df['patient_id'].apply(lambda x:
fake.random_element(elements=('Male', 'Female')))
kaggle_df['contact_info'] = kaggle_df['patient_id'].apply(lambda x: fake.address())

# Assign insurance_provider_id randomly
cursor.execute("SELECT insurance_provider_id FROM InsuranceProviders")
insurance_provider_ids = [row[0] for row in cursor.fetchall()]
kaggle_df['insurance_provider_id'] = kaggle_df['patient_id'].apply(lambda x:
fake.random_element(elements=insurance_provider_ids))

# Prepare patients DataFrame
patients_df = kaggle_df[['patient_id', 'date_of_birth', 'gender', 'contact_info',
'insurance_provider_id']].drop_duplicates(subset='patient_id')

for index, row in kaggle_df.iterrows():
    cursor.execute(
        """
```

# Final Project 1 Report

```
        INSERT INTO Patients (patient_id, date_of_birth, gender, contact_info,
insurance_provider_id)
        VALUES (%s, %s, %s, %s, %s)
        ON CONFLICT (patient_id) DO UPDATE
        SET date_of_birth = EXCLUDED.date_of_birth,
            gender = EXCLUDED.gender,
            contact_info = EXCLUDED.contact_info,
            insurance_provider_id = EXCLUDED.insurance_provider_id
        """,
        (
            row['patient_id'],
            row['date_of_birth'],
            row['gender'],
            row['contact_info'],
            row['insurance_provider_id'],
        )
    )
conn.commit()
print("Patients table populated successfully.")

### Transactional Tables

# Assign random admission_date
start_date = datetime(2020, 1, 1)
end_date = datetime(2020, 12, 31)
kaggle_df['admission_date'] = kaggle_df['patient_id'].apply(
    lambda x: fake.date_between_dates(date_start=start_date, date_end=end_date)
)

# Map 'diag_1' to 'reason_for_admission' using MedicalCodes
cursor.execute("SELECT code_value, description FROM MedicalCodes WHERE code_type =
'Diagnosis'")
code_mapping = {row[0]: row[1] for row in cursor.fetchall()}
kaggle_df['reason_for_admission'] = kaggle_df['diag_1'].map(code_mapping)

# Fetch department IDs
cursor.execute("SELECT department_id, department_name FROM HospitalDepartments")
departments = cursor.fetchall()
department_mapping = {dept_name: dept_id for dept_id, dept_name in departments}
```

# Final Project 1 Report

```
# Map 'medical_specialty' to 'department_id'
kaggle_df['department_id'] = kaggle_df['medical_specialty'].map(department_mapping)

# Fetch provider IDs
cursor.execute("SELECT provider_id, department_id FROM HealthcareProviders")
providers = cursor.fetchall()
providers_df = pd.DataFrame(providers, columns=['provider_id', 'department_id'])

# Function to assign provider_id based on department_id
def assign_provider(dept_id):
   available_providers = providers_df[providers_df['department_id'] ==
dept_id]['provider_id'].tolist()
   if available_providers:
       return fake.random_element(elements=available_providers)
   else:
       return None

kaggle_df['provider_id'] = kaggle_df['department_id'].apply(assign_provider)

# Prepare admissions DataFrame
admissions_df = kaggle_df[['patient_id', 'admission_date', 'reason_for_admission',
'department_id', 'provider_id']]
for index, row in admissions_df.iterrows():
   # Handle NaN or None values in the relevant columns
   reason_for_admission = row['reason_for_admission'] if
pd.notnull(row['reason_for_admission']) else 'Unknown'
   department_id = row['department_id'] if pd.notnull(row['department_id']) else None
   provider_id = row['provider_id'] if pd.notnull(row['provider_id']) else None
   # Insert into Admissions, omitting admission_id
   cursor.execute(
       """
       INSERT INTO Admissions (patient_id, admission_date, reason_for_admission,
department_id, provider_id)
       VALUES (%s, %s, %s, %s, %s)
       """,
       (
           row['patient_id'],
           row['admission_date'],
           reason_for_admission,
           department_id,
```

# Final Project 1 Report

```
            provider_id,
        )
    )

conn.commit()

print("Admissions table populated successfully.")
# Fetch admission_id and patient_id from the Admissions table
cursor.execute("SELECT admission_id, patient_id FROM Admissions")
admissions_from_db = cursor.fetchall()

# Create a mapping between patient_id and admission_id
admission_id_mapping = {row[1]: row[0] for row in admissions_from_db}

# Map admission_id back to kaggle_df based on patient_id
kaggle_df['admission_id'] = kaggle_df['patient_id'].map(admission_id_mapping)

# Ensure default values for department_id and provider_id
kaggle_df['reason_for_admission'].fillna('Unknown', inplace=True)
kaggle_df['department_id'].fillna(1, inplace=True) # Assuming 1 as the default
department
kaggle_df['provider_id'].fillna(1, inplace=True)   # Assuming 1 as the default
provider

# Update Admissions table with the correct values
for index, row in kaggle_df.iterrows():
    cursor.execute(
        """
        UPDATE Admissions
        SET reason_for_admission = %s,
            department_id = %s,
            provider_id = %s
        WHERE admission_id = %s
        """,
        (
            row['reason_for_admission'],
            row['department_id'],
            row['provider_id'],
            row['admission_id']
        )
```

# Final Project 1 Report

```
    )

conn.commit()
print("Admissions table updated successfully.")

## Discharge Table

from datetime import timedelta

# Calculate discharge_date by adding 'time_in_hospital' days to 'admission_date'
kaggle_df['discharge_date'] = kaggle_df.apply(
    lambda row: row['admission_date'] + timedelta(days=row['time_in_hospital']), axis=1
)

# Generate 'discharge_disposition' and 'follow_up_instructions' synthetically
discharge_dispositions = ['Discharged to home', 'Transferred to another hospital',
'Left against medical advice']
kaggle_df['discharge_disposition'] = kaggle_df['patient_id'].apply(
    lambda x: fake.random_element(elements=discharge_dispositions)
)
kaggle_df['follow_up_instructions'] = 'Follow up with primary care physician in 2
weeks.'

# Prepare discharges DataFrame
discharges_df = kaggle_df[['discharge_date', 'discharge_disposition',
'follow_up_instructions']]

# Fetch admission_ids from Admissions table
cursor.execute("SELECT admission_id, patient_id, admission_date FROM Admissions")
admissions_records = cursor.fetchall()
admissions_df_db = pd.DataFrame(admissions_records, columns=['admission_id',
'patient_id', 'admission_date'])

# Merge admission_ids back into discharges_df
discharges_df = admissions_df_db.merge(
    discharges_df,
    left_index=True,
    right_index=True
)
```

# Final Project 1 Report

```
# Insert data into Discharges table without specifying discharge_id
for index, row in discharges_df.iterrows():
   cursor.execute(
       """
       INSERT INTO Discharges (admission_id, discharge_date, discharge_disposition,
follow_up_instructions)
       VALUES (%s, %s, %s, %s)
       ON CONFLICT (admission_id) DO NOTHING
       """,
       (
           int(row['admission_id']),
           row['discharge_date'],
           row['discharge_disposition'],
           row['follow_up_instructions']
       )
   )

conn.commit()
print("Discharges table populated successfully.")

import pandas as pd
from datetime import timedelta

# Prepare procedure records
procedure_records = []

# For each admission, I created procedure records based on 'diag_1', 'diag_2',
'diag_3'
for index, row in kaggle_df.iterrows():
   admission_id = row['admission_id']
   provider_id = row['provider_id']
   procedure_date = row['admission_date'] + timedelta(days=1)  # Assume procedure is
done the next day after admission
   procedure_outcome = 'Successful'  # Assuming all procedures are successful

   # Iterate over diagnosis columns to assign procedure codes
   for diag_col in ['diag_1', 'diag_2', 'diag_3']:
       code_value = row[diag_col]
       if pd.notnull(code_value):
```

# Final Project 1 Report

```
            cursor.execute(
                """
                SELECT code_id FROM MedicalCodes WHERE code_value = %s
                """,
                (code_value,)
            )
            result = cursor.fetchone()

            if result is not None:
                code_id = result[0]
                procedure_records.append({
                    'admission_id': admission_id,
                    'provider_id': provider_id,
                    'procedure_date': procedure_date,
                    'code_id': code_id,
                    'procedure_outcome': procedure_outcome
                })
            else:
                print(f"Warning: No code_id found for diagnosis {code_value}")

# Convert the records into a DataFrame
procedures_df = pd.DataFrame(procedure_records)

# Insert data into MedicalProcedures table
for index, row in procedures_df.iterrows():
    # Skip the row if provider_id is NaN
    if pd.isna(row['provider_id']):
        print(f"Skipping row {index} due to missing provider_id.")
        continue

    cursor.execute(
        """
        INSERT INTO MedicalProcedures (admission_id, provider_id, procedure_date,
code_id, procedure_outcome)
        VALUES (%s, %s, %s, %s, %s)
        """,
        (
            int(row['admission_id']),
            int(row['provider_id']),  # Ensure it's a valid integer
            row['procedure_date'],
```

# Final Project 1 Report

```
            int(row['code_id']),
            row['procedure_outcome']
        )
    )

conn.commit()

print("MedicalProcedures table populated successfully.")



# Generate and insert Medications Administered data

medications_administered_records = []

for index, row in kaggle_df.iterrows():
    admission_id = row['admission_id']
    medication_id = fake.random_int(min=1, max=6)  # Assuming there are 6 medications
    dosage = f"{fake.random_int(min=1, max=2)} tablet(s)"  # Generate dosage as 1 or 2
tablets
    route_of_administration = 'Oral'  # Assuming oral route for simplicity
    administration_date = row['admission_date'] + timedelta(days=1)  # Day after
admission

    medications_administered_records.append({
        'admission_id': admission_id,
        'medication_id': medication_id,
        'dosage': dosage,
        'route_of_administration': route_of_administration,
        'administration_date': administration_date
    })

# Insert into MedicationsAdministered table
for record in medications_administered_records:
    cursor.execute(
        """
        INSERT INTO MedicationsAdministered (admission_id, medication_id, dosage,
route_of_administration, administration_date)
        VALUES (%s, %s, %s, %s, %s)
        """,
        (
```

# Final Project 1 Report

```
            record['admission_id'],
            record['medication_id'],
            record['dosage'],
            record['route_of_administration'],
            record['administration_date']
        )
    )

conn.commit()
print("MedicationsAdministered table populated successfully.")



# Generate and insert Laboratory Tests data

lab_tests_records = []

for index, row in kaggle_df.iterrows():
    admission_id = row['admission_id']
    test_type = fake.random_element(elements=('Glucose', 'A1C', 'Blood Panel', 'Lipid
Panel'))
    test_code = fake.random_int(min=1000, max=9999)  # Assuming test codes are random
integers
    test_date = row['admission_date'] + timedelta(days=2)  # Lab test done 2 days after
admission
    result = fake.random_element(elements=('Normal', 'Abnormal'))

    lab_tests_records.append({
        'admission_id': admission_id,
        'test_type': test_type,
        'test_code': test_code,
        'test_date': test_date,
        'result': result
    })

# Insert into LaboratoryTests table
for record in lab_tests_records:
    cursor.execute(
        """
        INSERT INTO LaboratoryTests (admission_id, test_type, test_code, test_date,
result)
```

# Final Project 1 Report

```
        VALUES (%s, %s, %s, %s, %s)
        """,
        (
            record['admission_id'],
            record['test_type'],
            record['test_code'],
            record['test_date'],
            record['result']
        )
    )

conn.commit()
print("LaboratoryTests table populated successfully.")

# Generate and insert Readmission Records data

readmission_records = []

for index, row in kaggle_df.iterrows():
    original_admission_id = row['admission_id']
    readmission_date = row['admission_date'] + timedelta(days=fake.random_int(min=30,
max=100))  # Readmission happens 30-100 days later
    readmission_reason = fake.sentence(nb_words=6)  # Random sentence for readmission
reason
    associated_costs = fake.random_int(min=500, max=5000)

    readmission_records.append({
        'original_admission_id': original_admission_id,
        'readmission_date': readmission_date,
        'readmission_reason': readmission_reason,
        'associated_costs': associated_costs
    })

# Insert into ReadmissionRecords table
for record in readmission_records:
    cursor.execute(
        """
        INSERT INTO ReadmissionRecords (original_admission_id, readmission_date,
readmission_reason, associated_costs)
        VALUES (%s, %s, %s, %s)
```

# Final Project 1 Report

```
        """,
        (
            record['original_admission_id'],
            record['readmission_date'],
            record['readmission_reason'],
            record['associated_costs']
        )
    )

conn.commit()
print("ReadmissionRecords table populated successfully.")

# Generate and insert Billing Records data

billing_records = []

for index, row in kaggle_df.iterrows():
    admission_id = row['admission_id']
    total_charges = fake.random_int(min=5000, max=20000)  # Random total charge between
5000 and 20000
    claim_status = fake.random_element(elements=('Pending', 'Approved', 'Denied'))
    patient_out_of_pocket = fake.random_int(min=500, max=3000)

    billing_records.append({
        'admission_id': admission_id,
        'total_charges': total_charges,
        'claim_status': claim_status,
        'patient_out_of_pocket': patient_out_of_pocket
    })

# Insert into BillingRecords table
for record in billing_records:
    cursor.execute(
        """
        INSERT INTO BillingRecords (admission_id, total_charges, claim_status,
patient_out_of_pocket)
        VALUES (%s, %s, %s, %s)
        """,
        (
            record['admission_id'],
```

# Final Project 1 Report

```
            record['total_charges'],
            record['claim_status'],
            record['patient_out_of_pocket']
        )
    )

conn.commit()
print("BillingRecords table populated successfully.")



# Generate and insert Patient Feedback data

feedback_records = []

for index, row in kaggle_df.iterrows():
    admission_id = row['admission_id']
    satisfaction_score = fake.random_int(min=1, max=5)  # Satisfaction score from 1 to
5
    comments = fake.sentence(nb_words=10)  # Random comment

    feedback_records.append({
        'admission_id': admission_id,
        'satisfaction_score': satisfaction_score,
        'comments': comments
    })

# Insert into PatientFeedback table
for record in feedback_records:
    cursor.execute(
        """
        INSERT INTO PatientFeedback (admission_id, satisfaction_score, comments)
        VALUES (%s, %s, %s)
        """,
        (
            record['admission_id'],
            record['satisfaction_score'],
            record['comments']
        )
    )
```

# Final Project 1 Report

```
conn.commit()
print("PatientFeedback table populated successfully.")



# Generate and insert Staff Scheduling data

staff_scheduling_records = []

for index, row in providers_df.iterrows():
    provider_id = row['provider_id']
    shift_date = fake.date_between(start_date='-30d', end_date='today')  # Last 30 days
    shift_start_time = fake.time()
    shift_end_time = fake.time()
    department_id = row['department_id']

    staff_scheduling_records.append({
        'provider_id': provider_id,
        'shift_date': shift_date,
        'shift_start_time': shift_start_time,
        'shift_end_time': shift_end_time,
        'department_id': department_id
    })
# Insert into StaffScheduling table

from datetime import datetime

staff_scheduling_records = []

for index, row in providers_df.iterrows():
    provider_id = int(row['provider_id'])
    department_id = row['department_id']

    # Skip if department_id is NaN
    if pd.isna(department_id):
        continue

    department_id = int(department_id)

    shift_date = fake.date_between(start_date='-30d', end_date='today')  # Last 30 days
```

# Final Project 1 Report

```
    # Generate times and ensure proper format
    shift_start_time_str = fake.time()
    shift_end_time_str = fake.time()

    shift_start_time = datetime.strptime(shift_start_time_str, '%H:%M:%S').time()
    shift_end_time = datetime.strptime(shift_end_time_str, '%H:%M:%S').time()

    staff_scheduling_records.append({
        'provider_id': provider_id,
        'shift_date': shift_date,
        'shift_start_time': shift_start_time,
        'shift_end_time': shift_end_time,
        'department_id': department_id
    })

for record in staff_scheduling_records:
    try:
        cursor.execute(
            """
            INSERT INTO StaffScheduling (provider_id, shift_date, shift_start_time,
shift_end_time, department_id)
            VALUES (%s, %s, %s, %s, %s)
            """,
            (
                record['provider_id'],
                record['shift_date'],
                record['shift_start_time'],
                record['shift_end_time'],
                record['department_id']
            )
        )
        conn.commit()
    except Exception as e:
        print(f"Error inserting StaffScheduling record: {record}")
        print(f"Exception: {e}")
        conn.rollback()
print("Staff Scheduling table populated successfully.")

#equipment table
```

# Final Project 1 Report

```
equipment_items = []

for equipment_id in range(1, 11):  # Equipment IDs from 1 to 10
   equipment_name = f"Equipment {equipment_id}"
   equipment_type = fake.random_element(elements=('MRI Machine', 'CT Scanner', 'X-Ray
Machine', 'Ultrasound', 'ECG Machine'))
   maintenance_schedule = fake.date_between(start_date='today', end_date='+30d')
   department_id = fake.random_int(min=1, max=10)  # Assuming department IDs from 1 to
10

   equipment_items.append({
       'equipment_id': equipment_id,
       'equipment_name': equipment_name,
       'equipment_type': equipment_type,
       'maintenance_schedule': maintenance_schedule,
       'department_id': department_id
   })


# Insert into Equipment table
for item in equipment_items:
   cursor.execute(
       """
       SELECT COUNT(*) FROM Equipment WHERE equipment_id = %s
       """,
       (item['equipment_id'],)
   )
   count = cursor.fetchone()[0]

   if count == 0:  # Only insert if the equipment_id doesn't already exist
       cursor.execute(
           """
           INSERT INTO Equipment (equipment_id, equipment_name, equipment_type,
maintenance_schedule, department_id)
           VALUES (%s, %s, %s, %s, %s)
           """,
           (
               item['equipment_id'],
               item['equipment_name'],
               item['equipment_type'],
```

# Final Project 1 Report

```
                item['maintenance_schedule'],
                item['department_id']
            )
        )
    else:
        print(f"Equipment ID {item['equipment_id']} already exists. Skipping
insertion.")
conn.commit()
print("Equipment table populated successfully.")
```

### Milestone 3:

### Multidimensional Model for Hospital Readmissions Data Warehouse

The multidimensional model for the hospital readmissions data warehouse is structured using a star schema, which is particularly well-suited for analytical queries in OLAP systems. The model is designed to support the analysis of hospital readmissions data, with the goal of improving patient care by identifying trends and factors that contribute to readmissions.

- Fact Table:
    - The Readmission Facts table serves as the core of the model, storing key performance metrics such as total charges, length of stay, satisfaction scores, and readmission rates. This table links to several dimensions, enabling comprehensive analysis of patient readmission patterns.
- Dimension Tables:
    - Patient Dimension: Contains demographic information such as age group, gender, ethnicity, and insurance provider details, allowing analysis based on patient characteristics.
    - Time Dimension: Facilitates temporal analysis with attributes like day, month, quarter, and year.
    - Provider Dimension: Captures details about healthcare providers, including their specialty and associated department.
    - Department Dimension: represents hospital departments, including their location within the facility.
    - Procedure Dimension: Stores medical procedure codes and descriptions to enable analysis of the types of treatments associated with admissions.
    - Insurance Dimension: Holds information about the insurance providers involved in patient care.

This multidimensional model allows for complex OLAP operations, such as slicing data by patient demographics, dicing across time and departments, performing roll-ups and drill-downs

# Final Project 1 Report

through time hierarchies, and pivoting data to compare performance across departments or providers. These capabilities are critical for identifying patterns that can inform strategies to reduce hospital readmissions and enhance patient outcomes.

This fact table is designed to support key performance indicators and analyses by tracking the following measures:

Measures:

- Number of Readmissions
- Total Charges
- Satisfaction Scores
- Associated Costs

Cardinality:

- A patient can have multiple readmissions.
- A provider can perform multiple procedures during an admission.
- The time dimension is linked to many admissions and discharges, supporting historical and trend-based analysis.

This multidimensional model, structured with these fact and dimension tables, allows for in-depth analysis of hospital readmissions, enabling stakeholders to identify patterns and trends that can help improve patient outcomes and reduce operational costs.

### Conceptual Model for Hospital Readmissions Data Warehouse

Entities and Relationships

- Core Entities:
    a. Patient: Represents individuals admitted to the hospital.
    b. Provider: Represents healthcare providers responsible for patient care.
    c. Time: Represents dates and time periods related to admissions and discharges.
    d. Diagnosis: Represents the primary medical conditions associated with readmissions.
    e. Location: Represents location-related information for patient and provider distribution analysis.
    f. Insurance Provider: Represents the patient's insurance provider, impacting cost and readmission trends.
    g. Department: Represents hospital departments where care is provided, e.g., cardiology, emergency.
    h. Readmission Facts: The central entity capturing each readmission event, aggregating information from all relevant dimensions.
    i. Admission Facts: Another central entity capturing each admission event, aggregating information from all relevant dimensions.
- Relationships:

# Final Project 1 Report

a. A Patient is admitted by a Provider to a specific Department and has Insurance coverage.
b. A Diagnosis is assigned to each Readmission Fact event, which may include associated Medications.
c. Readmission Facts are time-stamped by Time dimensions to allow tracking over specific periods.
d. Geographic dimensions relate to both Patient and Provider for location-based analytics.

Key attributes for each entity:

- Patient: Age group, gender, ethnicity.
- Provider: Provider name, specialty, department affiliation.
- Time: Date ID, day, month, quarter, year.
- Diagnosis: Code value, description.
- Geographic: City, state, region.
- Insurance Provider: Provider name, contact information.
- Department: Department name, location.
- Medication: Medication name, dosage form, administration route.
- Readmission Facts: Total charges, average length of stay, satisfaction score, associated costs.

Hierarchies

- Time: Supports drill-down through Day > Month > Quarter > Year for trends and seasonal analysis.
- Patient: Facilitates segmentation through Age Group > Gender > Ethnicity for demographic insights.
- Location: Enables regional analysis through City > State > Region to assess geographic patterns.

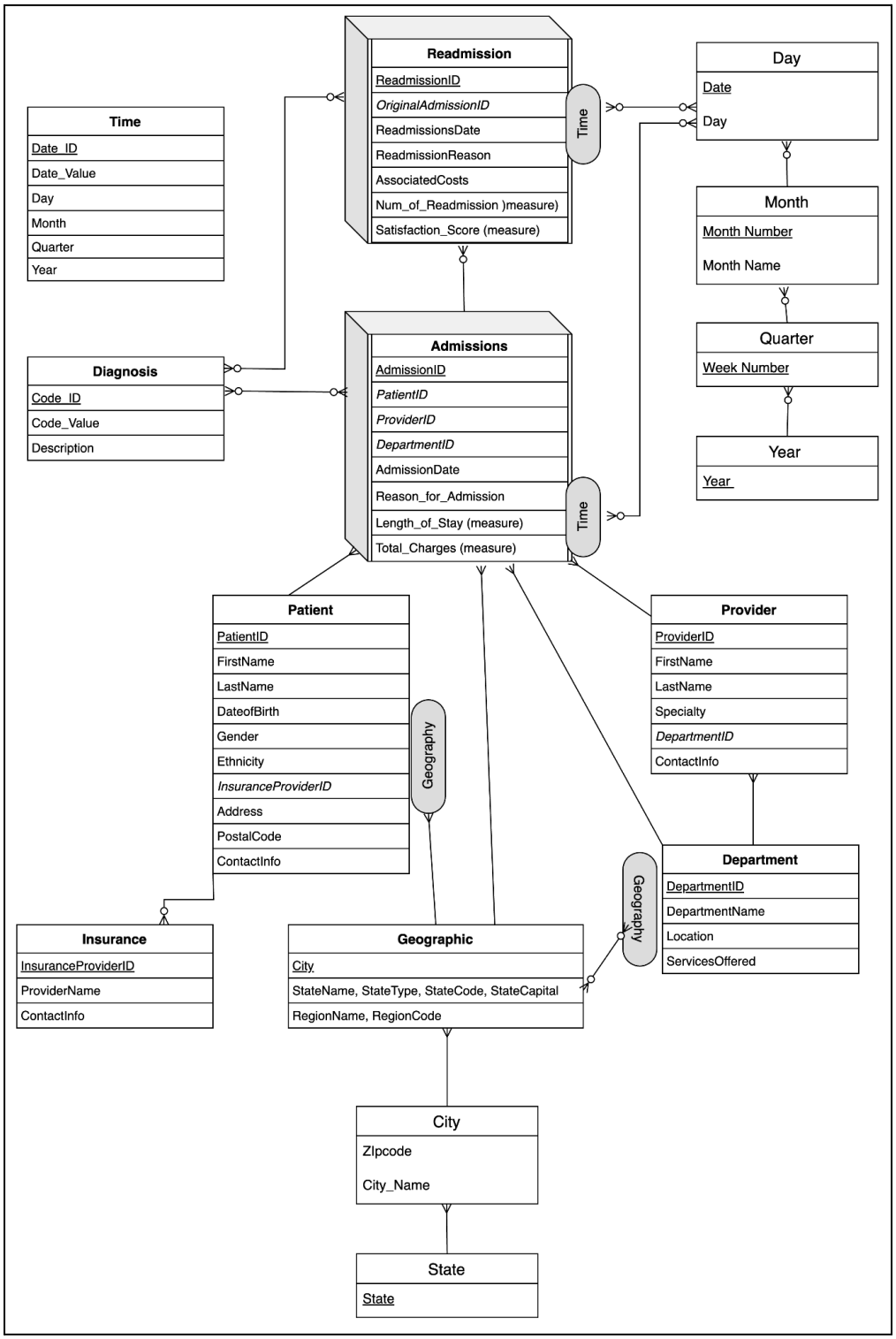
High-Level Business Processes

- Readmission Trend Analysis: Tracks readmission rates and identifies patterns in time and across departments.
- Cost and Revenue Analysis: Examines associated costs and charges by readmission events, insurance providers, and departments.
- Provider Performance: Assesses satisfaction scores and readmission rates by provider and department.
- Patient Demographic Analysis: Analyzes readmission trends across different patient demographics and geographic areas.
- Medication Efficacy: Monitors medication effectiveness by diagnosis and readmission rates.

### Conceptual Data Warehouse (DW) Model

The Conceptual Data Warehouse Model for this project is as below:

# Final Project 1 Report

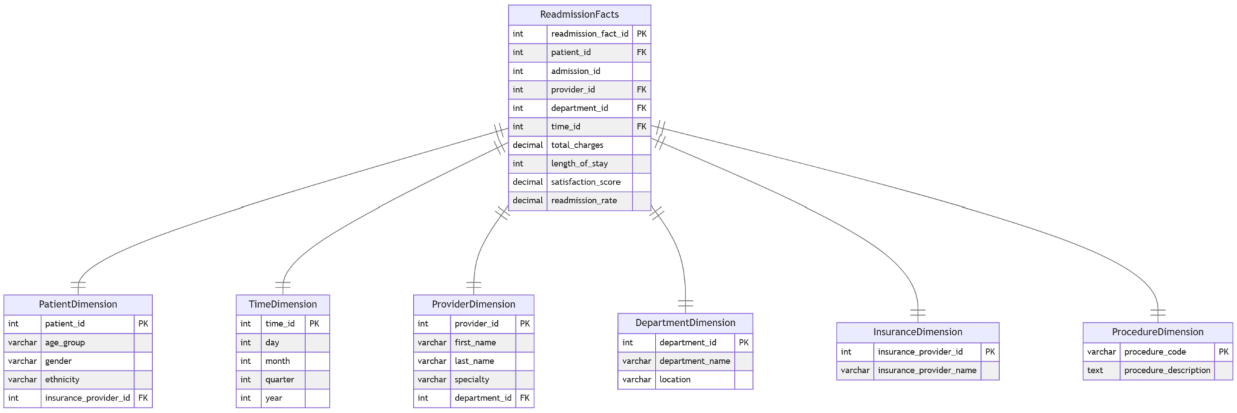


# Final Project 1 Report

## Logical Data Warehouse (DW) Model

This step is crucial for the PostgreSQL implementation and ensures that the relational model is logically sound and can support OLAP queries.

The logical data warehouse model maps the conceptual multidimensional model into a relational schema. This includes defining all tables, primary keys (PKs), foreign keys (FKs), references, and their relationships.




**Fact Table: patient_readmissions**:

```
CREATE TABLE patient_readmissions (
    readmission_id SERIAL PRIMARY KEY,
    patient_id INTEGER REFERENCES patient_dimension(patient_id),
    provider_id INTEGER REFERENCES provider_dimension(provider_id),
    admission_date INTEGER REFERENCES time_dimension(date_id),
    discharge_date INTEGER REFERENCES time_dimension(date_id),
    code_id INTEGER REFERENCES diagnosis_dimension(code_id),
    total_charges DECIMAL,--measure
    average_length_of_stay DECIMAL,--measure
    satisfaction_score INTEGER, —-measure
    associated_costs DECIMAL -- measure
);
```

**Dimension Tables**:

- **Patient Dimension**:

```
    CREATE TABLE patient_dimension (
        patient_id SERIAL PRIMARY KEY,
        age_group VARCHAR(20),
```

# Final Project 1 Report

```
        gender VARCHAR(10),
        ethnicity VARCHAR(50),
        insurance_provider_id INTEGER
);
```

- **Time Dimension**:

```
CREATE TABLE time_dimension (
        date_id SERIAL PRIMARY KEY,
        day INTEGER,
        month INTEGER,
        quarter INTEGER,
        year INTEGER
);
```

- **Provider Dimension**

```
CREATE TABLE provider_dimension (
        provider_id SERIAL PRIMARY KEY,
        specialty VARCHAR(100),
        department_id INTEGER
);
```

- **Diagnosis Dimension**:

```
CREATE TABLE diagnosis_dimension (
        code_id SERIAL PRIMARY KEY,
        code_value VARCHAR(20),
        description TEXT
);
```

- **Geographic Dimension**:

```
CREATE TABLE geographic_dimension (
        location_id SERIAL PRIMARY KEY,
        city VARCHAR(50),
        state VARCHAR(50),
```

# Final Project 1 Report

```
            region VARCHAR(50)
      );
```

This logical model maps directly from the conceptual model to relational tables in PostgreSQL, ensuring the correct structure for OLAP queries.

### Database Implementation

The implementation of the relational schema for the hospital readmissions data warehouse was carried out in PostgreSQL. This schema adheres to the star schema design and includes a central fact table connected to multiple dimension tables through well-defined primary and foreign keys. The implementation aims to meet the "Exceeds Standard" criteria by enforcing data integrity using constraints and establishing the necessary relationships between tables to support efficient querying and analysis. Below is a detailed explanation of the steps taken to implement the database.

To ensure proper relational structure, each table was implemented with a primary key to uniquely identify each record. The fact table—which serves as the core for storing measurable events related to hospital readmissions—references all relevant dimension tables using foreign keys. This structure allows for efficient data analysis while maintaining referential integrity across the entire schema.

**Fact Table:** ReadmissionFacts

- The ReadmissionFacts table stores data related to each patient's readmission event. It includes foreign keys that reference various dimension tables, such as PatientDimension, TimeDimension, ProviderDimension, and others.
- The primary key for this table is readmission_fact_id, which uniquely identifies each record in the fact table. Foreign keys are used to connect to the relevant dimension tables.

**Dimension Tables:** Each dimension table contains a primary key that uniquely identifies records within the dimension. These keys are referenced by the fact table to maintain the integrity of the schema. For example, the PatientDimension table contains patient information and is linked to the fact table via the patient_id foreign key.

**Constraints:** To maintain data accuracy and ensure the integrity of the data within each table, several constraints were implemented:

- NOT NULL: This constraint ensures that fields essential to analysis, such as total_charges and length_of_stay, cannot be left empty. This guarantees that key measures are always available for analytical queries.
- CHECK: This constraint ensures that values meet specific criteria, such as the satisfaction score being within a valid range (1 to 5) and length of stay being a positive value.

# Final Project 1 Report

- Referential Integrity: All foreign keys were defined with REFERENCES constraints to enforce the relationship between the fact table and the dimension tables. This prevents orphaned records in the fact table, ensuring that every fact has associated dimension data.

### Implementation of Fact and Dimension Tables

In the PostgreSQL implementation, each fact and dimension table was created with proper SQL syntax to ensure accurate data modeling. Below is a brief overview of the key tables and their structures.

This Patients Readmissions table tracks all relevant events associated with hospital readmissions, such as the total charges, length of stay, satisfaction scores, and associated costs. The table references multiple dimension tables through foreign keys, ensuring that each readmission is contextualized with patient, provider, time, diagnosis, and geographic data.

Dimension tables such as PatientDimension, TimeDimension, ProviderDimension, DiagnosisDimension, and GeographicDimension provide the necessary descriptive context for analyzing the events stored in the fact table. These tables store attributes like patient demographics, time hierarchies, provider information, diagnostic codes, and geographic data.

## Primary Events

The key business events tracked in the system are represented in the fact table. These events include Patient Readmission, Medication Administered, and Billing Records, each of which plays a major role in understanding and analyzing hospital readmissions.

- **Patient Readmission:** This is the primary event captured by the fact table. Each record tracks a specific readmission event for a patient, including details such as admission and discharge dates, length of stay, and associated costs. This event is central to the goal of analyzing and reducing hospital readmissions.
- **Medication Administered:** This event records the medications administered to patients during their readmission. By tracking this data, the system can help identify patterns between medication use and patient outcomes, informing future treatment decisions.
- **Billing Records:** Captures all financial data related to each readmission, including total charges, insurance claims, and patient out-of-pocket expenses. This data is essential for understanding the economic impact of readmissions on both the hospital and the patient.
- **Staff Scheduling and Provider Availability:** Tracks healthcare provider schedules, including shift details and provider availability during the patient's stay. Provider availability can impact patient outcomes, particularly in high-stress departments like Emergency. By correlating staff availability with readmission rates, the system can help identify potential staffing-related factors affecting patient care.

# Final Project 1 Report

- **Medical Procedures:** Records all medical procedures performed during a patient's stay, including details like procedure type, date, and outcomes. By analyzing the procedures associated with readmitted patients, the system can help identify which procedures might correlate with higher or lower readmission rates, informing future care protocols.

Each of these events is directly connected to the business goals of reducing readmissions and optimizing patient care, making them important parts of the overall data warehouse model.

### OLAP Operations

The OLAP operations outlined below can enable analysis in depth for healthcare administrators, facilitating data-driven decision-making to reduce readmissions and improve patient care. These operations leverage the multidimensional structure of the data warehouse to provide flexible, in-depth insights into key performance indicators.

- **Roll-Up**
  Objective: Aggregate data to a higher level of summary to monitor trends at a broader time interval.
  Example: ROLLUP*(Readmissions, AdmissionDate → Month, Department → Specialty, SUM(TotalCharges))
  Description: This operation aggregates daily readmission costs to monthly totals, allowing administrators to observe spending patterns over a larger time frame. This overview aids in budget management by highlighting monthly or quarterly cost trends.
- **Drill-Down**
  Objective: Access more granular details within specific departments or categories.
  Example: DRILLDOWN(Readmissions (Department → Specialty)
  Description: This drill-down operation enables administrators to break down hospital-wide readmission data by specialty, offering deeper insights into areas with higher readmission rates. By isolating data at the specialty level, administrators can target improvement efforts more effectively.
- **Slice**
  Objective: Filter data to focus on a specific condition or demographic.
  Example: SLICE(Readmissions, Diagnosis = 'Hypertension')
  Description: Slicing allows administrators to isolate readmission data for a particular diagnosis, such as hypertension. This operation supports targeted resource allocation and interventions by focusing on conditions with potentially higher readmission risks.
- **Dice**
  Objective: Filter data across multiple dimensions for a specific subgroup analysis.
  Example: DICE(Readmissions, [AgeGroup = '40-50', InsuranceProvider = 'Provider A'])
  Description: Dicing allows for multidimensional filtering. Here, it enables a focused analysis of patients aged 40-50 who are insured by Provider A, providing insights into readmission trends within this demographic and insurance category.
- **Pivot**
  Objective: View data from alternative perspectives to identify demographic trends.

# Final Project 1 Report

Example: PIVOT(Readmissions, (AgeGroup → X)
Description: This operation enables administrators to pivot the data, comparing readmission rates or costs across age groups and gender. Pivoting reveals demographic patterns, assisting in identifying trends in healthcare needs among different patient segments.

- **Drill-Across**
  Objective: Compare related data across different specialties or departments.
  Example: DRILLACROSS(Readmissions, Specialty, [Provider])
  Description: The drill-across operation provides comparative insights across providers within specialties. By analyzing readmission rates across these dimensions, administrators can identify differences in provider performance and determine where additional resources or support may be necessary.
- **Rename**
  Objective: To change the name following business requirements.
  Example: RENAME(HospitalReadmissionsCube,
  (PatientDimension.AgeGroup --> PatientAgeGroup),
  (ProviderDimension.Specialty --> ProviderSpecialization),
  (DepartmentDimension.DepartmentName --> Department)
  );
  Description: The rename operation in OLAP allows administrators to change the names based on the business needs.

By applying roll-up, drill-down, slice, dice, pivot, drill-across and rename, administrators gain insights that go beyond basic numbers. These operations reveal patterns, enable forecasting, and highlight areas for improvement, ultimately helping to make decisions that benefit both patients and the healthcare system as a whole.

# Final Project 1 Report

### Milestone 4

### Project Title: **Optimizing Patient Care: A Hospital Readmissions Database**

#### Details from Previous Milestones:

*Milestone 1: Project Initiation and Problem Definition*

The project titled "Optimizing Patient Care: A Hospital Readmissions Database" aims to address the high cost and operational impact of hospital readmissions. In this milestone, we defined the problem and identified key objectives such as reducing readmission rates, improving patient outcomes, and ensuring data privacy and security (HIPAA compliance). A Kaggle dataset was selected for analysis, and PostgreSQL was chosen to implement the operational database as part of the course requirement. The design includes tracking patient demographics, medical history, treatment data, and key performance indicators to support predictive analytics and efficient data management.

*Milestone 2: Data Modeling and Schema Design*

This milestone covered designing an Extended Entity-Relationship Diagram (ERD) and mapping it to a normalized relational model. Here, key data entities include patients, healthcare providers, admissions, medical procedures, and readmission records, among others. We also identified potential dimensions and measures for analytics, such as the patient, time, provider, and diagnosis dimensions. The model was implemented in PostgreSQL, establishing primary and foreign key constraints to ensure data integrity and support efficient OLAP operations.

*Milestone 3: Multidimensional Data Model and Data Warehouse Implementation*

In this milestone, we created a star schema-based data warehouse to support OLAP queries. We linked a core fact table for readmission metrics to several dimensions, including patient, provider, department, time, and procedure. We implemented OLAP operations such as slicing, dicing, roll-up, and drill-down to enable detailed analysis across patient demographics, treatment types, and healthcare provider performance. This setup allows healthcare administrators to identify trends, optimize patient care, and reduce readmissions through data-driven decision-making.

#### Stakeholder Impact

With fully operational tables, optimized ETL flows, and secure OLAP functionalities, this milestone marks a pivotal step in delivering actionable insights on hospital readmissions. This data warehouse provides a powerful tool for healthcare administrators, enabling data-driven decisions to reduce readmission rates and improve patient outcomes.

# Final Project 1 Report

## ETL Execution

### 1. Sourcing Data from the Operational Database

The primary data source for the data warehouse is the operational database designed in previous milestones. This database captures a wide range of hospital-related data, including patient admissions, readmissions, discharge details, medical procedures, billing records, and patient feedback. The operational database is updated with new transactions, and this data flows into the data warehouse. We used 2 data sources here and also synthesized data as well.

### 2. Data Warehouse Table Loading and Surrogate Key Implementation

We utilized the schema and detailed entity relationships established in Milestone 2, where core entities like patient demographics, departments, providers, and admissions were defined. The schema, sourced from a CSV file that updates patient dimensions, Kaggle and the Centers for Medicare & Medicaid Services as identified in Milestone 1, is now fully populated. Additional data was also artificially generated using Python.

### 3. Insert and Update Flows

Our ETL processes are configured to handle both inserts and updates, ensuring continuous data integrity and completeness for dynamic entities such as patient admissions and provider information. These flows were designed to meet the quality standards discussed in Milestone 1.

### 4. Dimensional and Fact Table Population with ETL Prepopulations

Leveraging the star schema structure from Milestone 3, the Readmission Facts table serves as the fact table, linking to prepopulated dimensions (Patient, Provider, Diagnosis, etc.) for efficient OLAP queries. This organization provides a streamlined data flow and supports optimal analysis.

### 5. Transformation Types and Additional Features

Our transformation approach is based on data volatility requirements identified in Milestone 2. For instance, patient demographics are managed through Type 1 transformations, while changes in insurance provider information are recorded as Type 2 Slowly Changing Dimensions (SCDs), preserving historical context.

### 6. Advanced Analytics, OLAP Operations, Future Analysis

Expanding on Milestone 3's star schema, the OLAP-enabled database supports advanced analytical capabilities. We incorporated roll-up operations (e.g., monthly to quarterly aggregations), drill-downs by department and provider specialty, and pivot views for cross-sectional comparisons across patient demographics.

1. Readmission Trends Analysis

- Objective: Identify patterns in patient readmissions to determine which demographics, departments, and diagnoses have the highest readmission rates.

# Final Project 1 Report

- Our Approach:
    - Aggregate readmissions by patient demographics (age, ethnicity, gender) and department.
    - Use time-series analysis to observe trends in readmission rates by diagnosis and department.
- Possible Insights: Highlight departments or demographic groups with higher-than-average readmission rates, supporting targeted interventions or resource allocation for these areas.

2. Cost Analysis by Diagnosis and Department

- Objective: Understand which diagnoses and departments incur the highest costs, supporting budget optimization and resource allocation.
- Our Approach:
    - Use the Avg_cost_by_diagnosis aggregation to identify high-cost conditions.
    - Break down total charges by department and diagnosis to compare against patient outcomes.
- Possible Insights: Pinpoint costly diagnoses and departments that could benefit from efficiency improvements or cost-saving initiatives.

3. Patient Satisfaction Analysis

- Objective: Assess patient satisfaction scores by provider and department to identify strengths and areas for improvement in patient experience.
- Our Approach:
    a. Analyze the Avg_satisfaction_by_provider metric to assess each provider's performance.
    b. Compare satisfaction scores by department and patient demographics to identify any trends or disparities.
- Possible Insights: Identify high-performing providers and departments in terms of patient satisfaction, which can help in replicating best practices across the hospital.

4. Insurance Impact on Readmissions and Costs

- Objective: Evaluate how different insurance providers affect readmission rates and treatment costs, which may indicate disparities in access to care or coverage quality.
- Our Approach:
    a. Track readmissions and treatment costs by insurance provider using SCD Type 2 historical data.
    b. Compare readmission rates and costs for different insurance plans over time.
- Possible Insights: Reveal patterns that may suggest how insurance type impacts patient outcomes and cost structures, potentially guiding hospital negotiations with insurers or highlighting areas for patient support.

5. Provider Performance Analysis by Department

- Objective: Evaluate provider performance across different departments based on patient outcomes and satisfaction.
- Our Approach:
    - Compare readmission rates and satisfaction scores by provider, broken down by department.

        - Identify any providers with consistently high or low performance across multiple metrics.
    - Possible Insights: Help identify high-performing providers who may be mentors or leaders within their departments, as well as those who may benefit from additional support.

6. Forecasting Readmission Rates

- Objective: Use historical data to forecast future readmission rates, enabling proactive planning for staffing and resource allocation.
- Our Approach:
    - Apply time-series forecasting techniques on readmission rates across departments and diagnoses.
    - Identify seasonal trends or fluctuations that could inform resource planning and preventive measures.
- Possible Insights: Anticipate potential surges in readmissions, allowing the hospital to adjust resources, improve discharge planning, or implement preventive interventions.

7. Correlation Analysis between Demographics and Outcomes

- Objective: Discover correlations between patient demographics (age, gender, ethnicity) and healthcare outcomes, such as readmission rates and satisfaction.
- Our Approach:
    - Use correlation analysis to assess relationships between demographics and outcomes.
    - Segment data by age group, ethnicity, or gender to understand variations in patient experiences or outcomes.
- Possible Insights: Gain an understanding of demographic factors impacting patient outcomes, guiding targeted interventions for specific groups.

### 7. Transformations

Transformations play a vital role in preparing the data for analytical queries. Two main types of transformations are applied:

- **Type 1 Transformations (Overwrites):** These transformations update non-historical attributes, such as patient contact information, directly overwriting old values with new ones.
- **Type 2 Transformations (Slowly Changing Dimensions):** These transformations track historical changes by creating new records for attributes that require versioning, such as changes in a patient's insurance provider. This approach preserves historical data, which is critical for trend analysis and reporting.

Please refer to the Appendix (ii. Talend Implementation, screenshot #3) to see further details.

### 8. Data Aggregations

Data aggregations are implemented to support high-level reporting and OLAP operations.

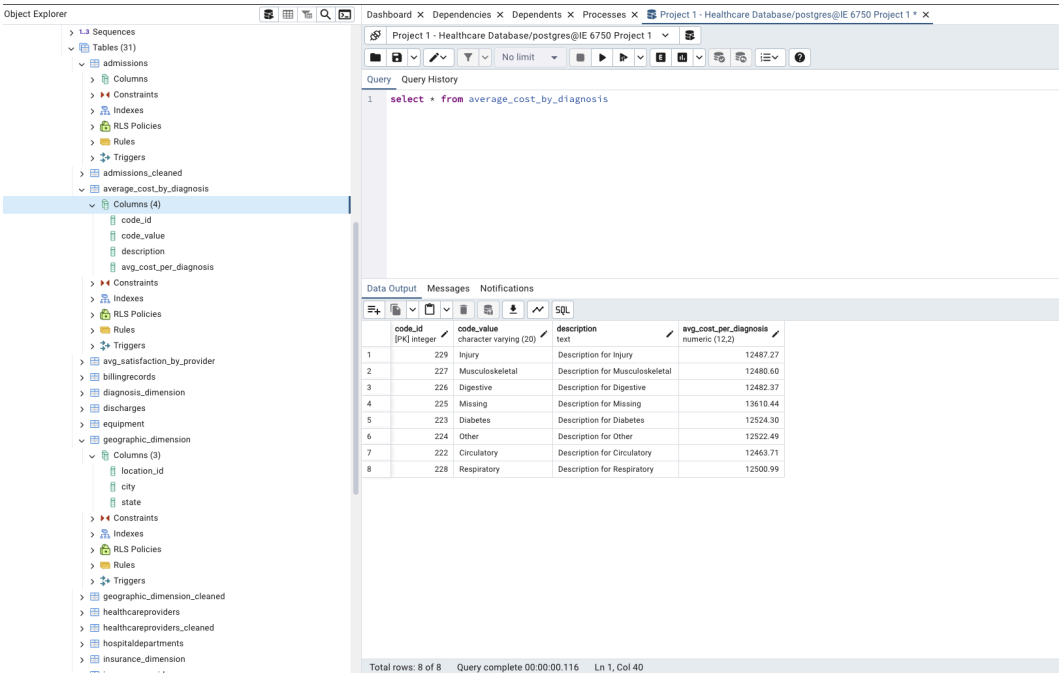
**1. Average Cost by Diagnosis (Avg_cost_by_diagnosis):** This aggregation calculates the average cost associated with each diagnosis in the dataset. This measure is essential for identifying the financial impact of various medical conditions on hospital

# Final Project 1 Report

resources. By examining costs across diagnoses, healthcare administrators can pinpoint which conditions are more resource-intensive. This insight can guide strategic planning, such as optimizing treatment protocols for high-cost diagnoses or allocating resources more effectively.

Calculation:

- *Input:* Records of patient admissions, including fields like DiagnosisCode and Cost
- *Output:* A summary table that shows each DiagnosisCode alongside an Avg_Cost, representing the mean cost associated with that particular diagnosis.

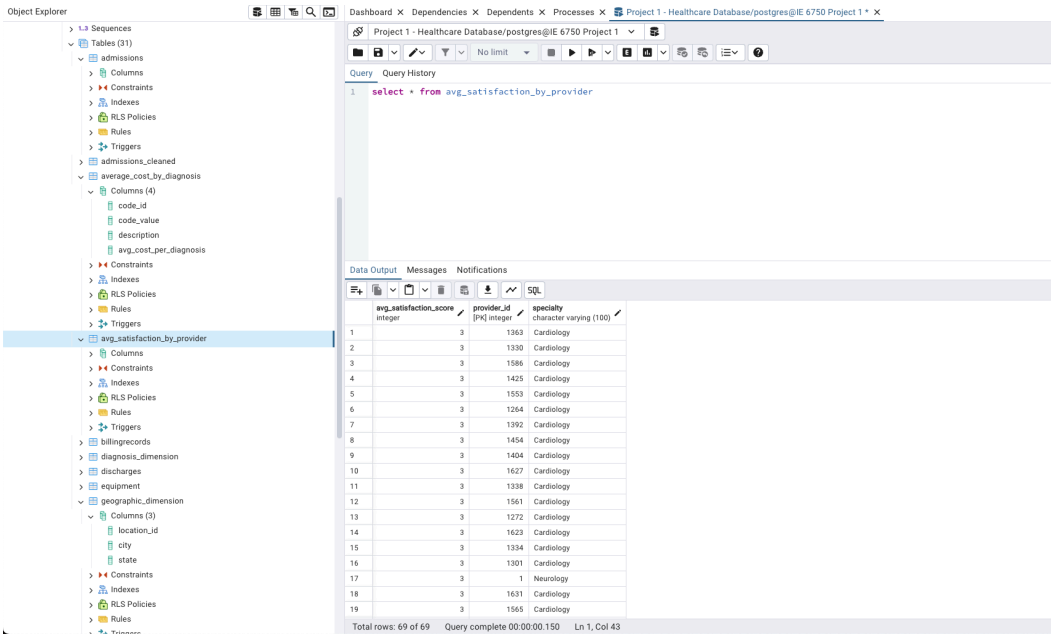



**2. Average Satisfaction by Provider (Avg_satisfaction_by_provider):** This aggregation provides the average patient satisfaction score for each healthcare provider. This metric offers a valuable perspective on provider performance from the patient's viewpoint. Consistently high satisfaction scores often reflect positive patient experiences, while lower scores can highlight areas where patient interactions, treatment approaches, or communication might need improvement.

Calculation:

- *Input*: Patient satisfaction data with fields like ProviderID and SatisfactionScore
- *Output*: A table that aggregates each ProviderID with an Avg_Satisfaction, showing the average satisfaction score attributed to each provider.

# Final Project 1 Report




### 9. Complete and Incremental Loads

To maintain an up-to-date data warehouse, we used two types of loading strategies:

- **Complete Loads:** A full load is performed during the initial setup of the hospital readmissions data warehouse or in rare cases when a total reload is necessary, such as after a major schema modification or when adding a new data source. This process ensures that all data is freshly loaded, providing a clean, comprehensive baseline for subsequent analysis and reporting.

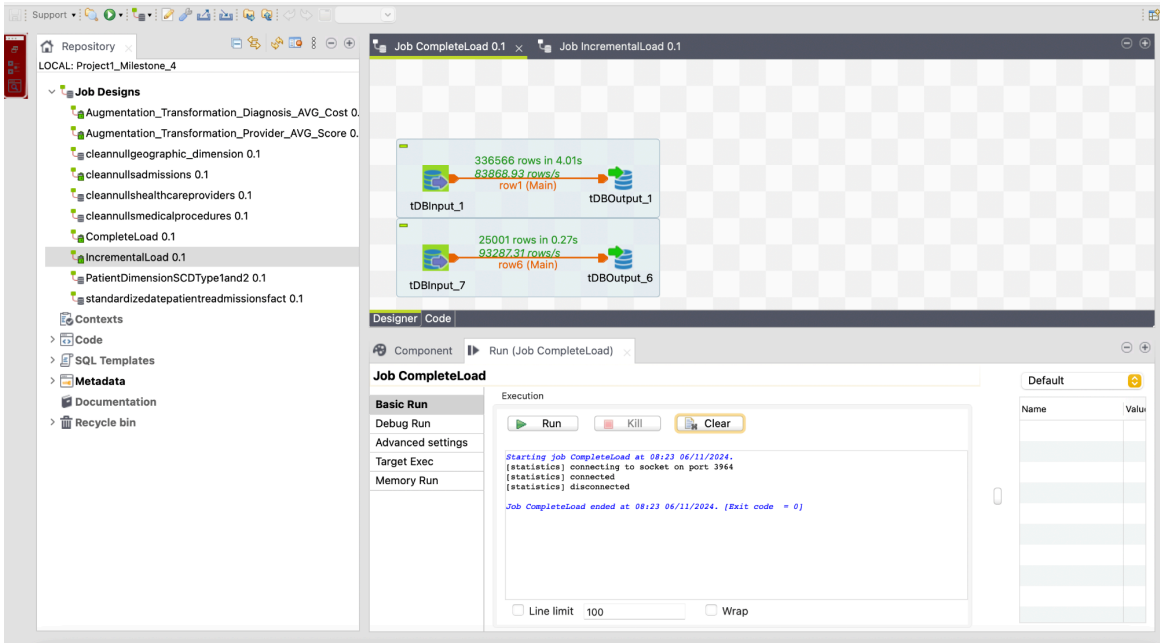



*Image: Complete Load Example in Talend.*

# Final Project 1 Report

- **Incremental Loads:** For ongoing updates, incremental loads capture only new or modified records since the previous load, using time-stamped changes in the hospital readmissions database. This ensures the data warehouse stays current with minimal data transfer and processing overhead.

  Changes to patient details or insurance information are updated incrementally, capturing only the modifications since the last ETL run.

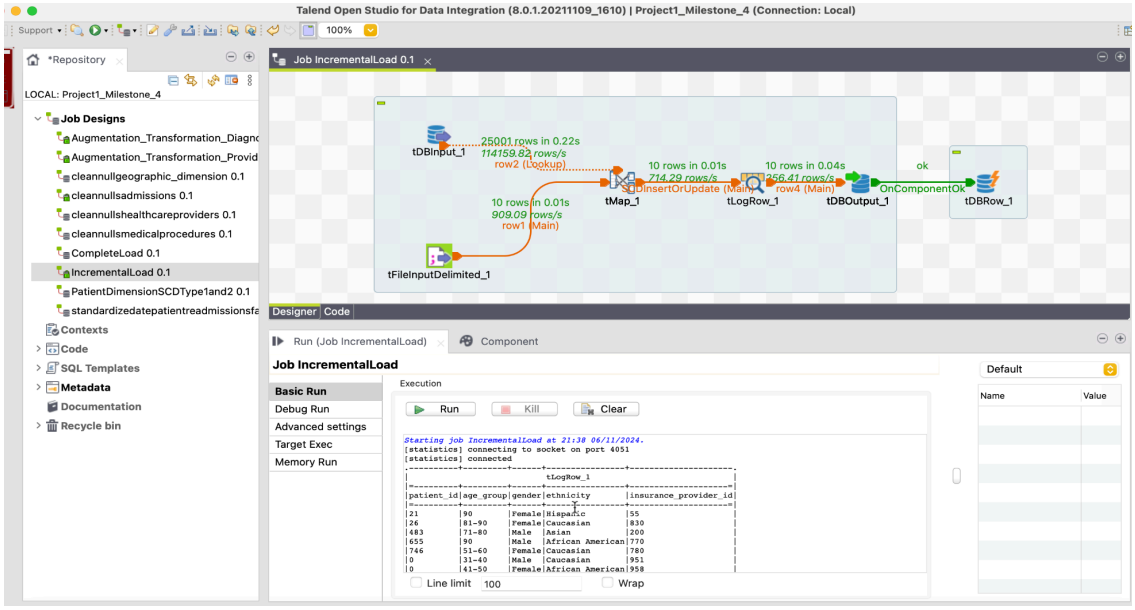
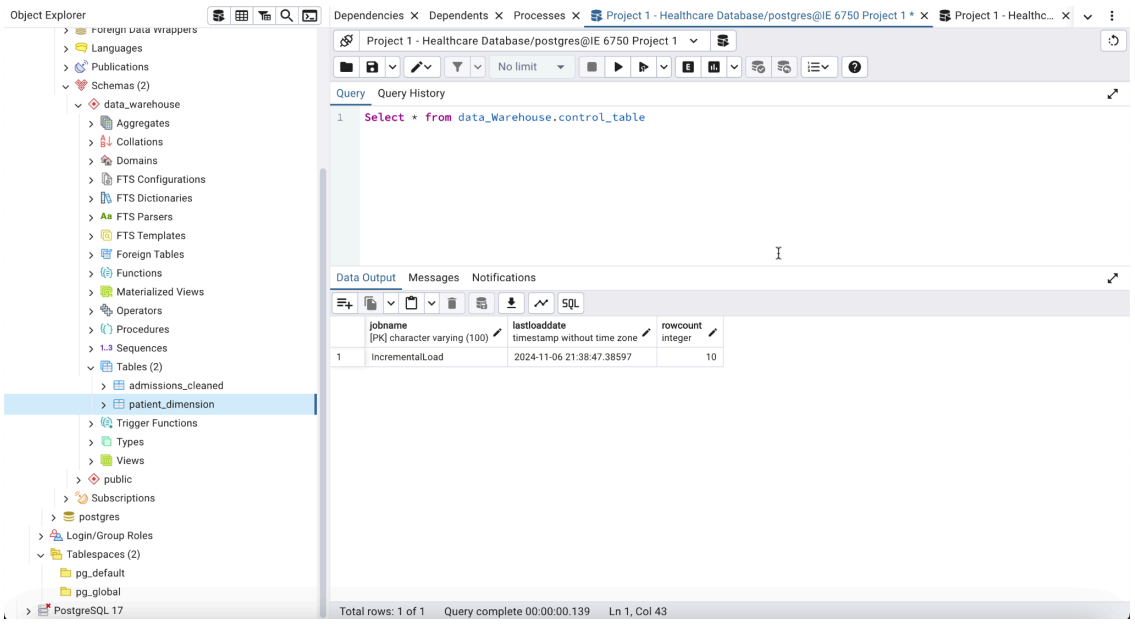



*Image: Incremental load example in Talend.*



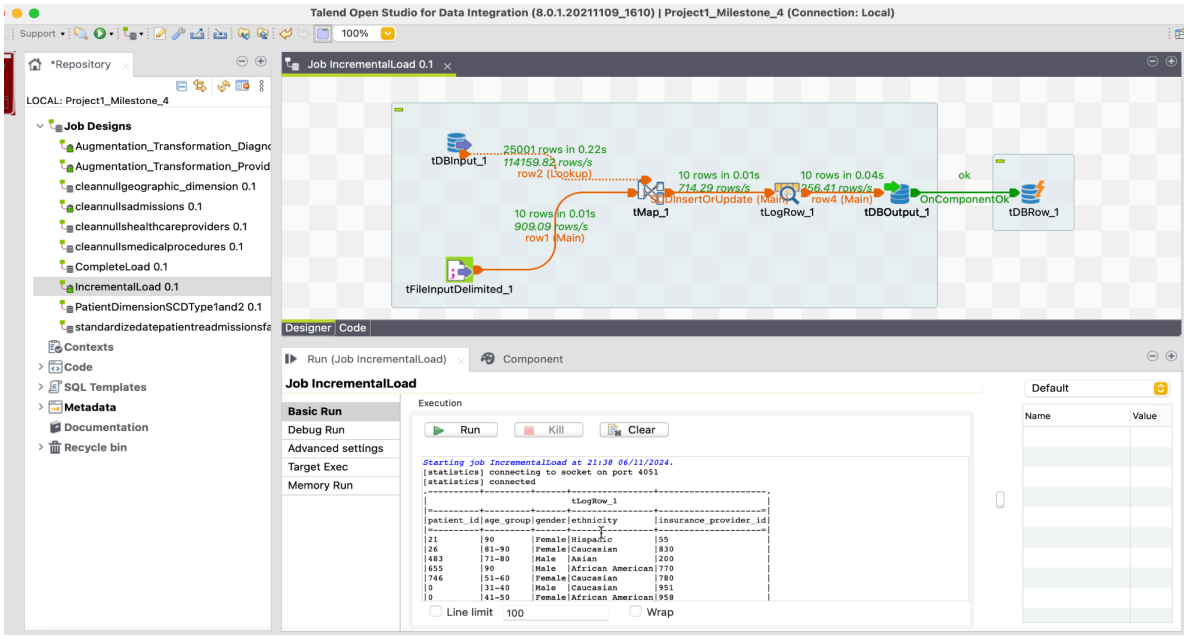
# Final Project 1 Report

### 10. Control Flow Documentation

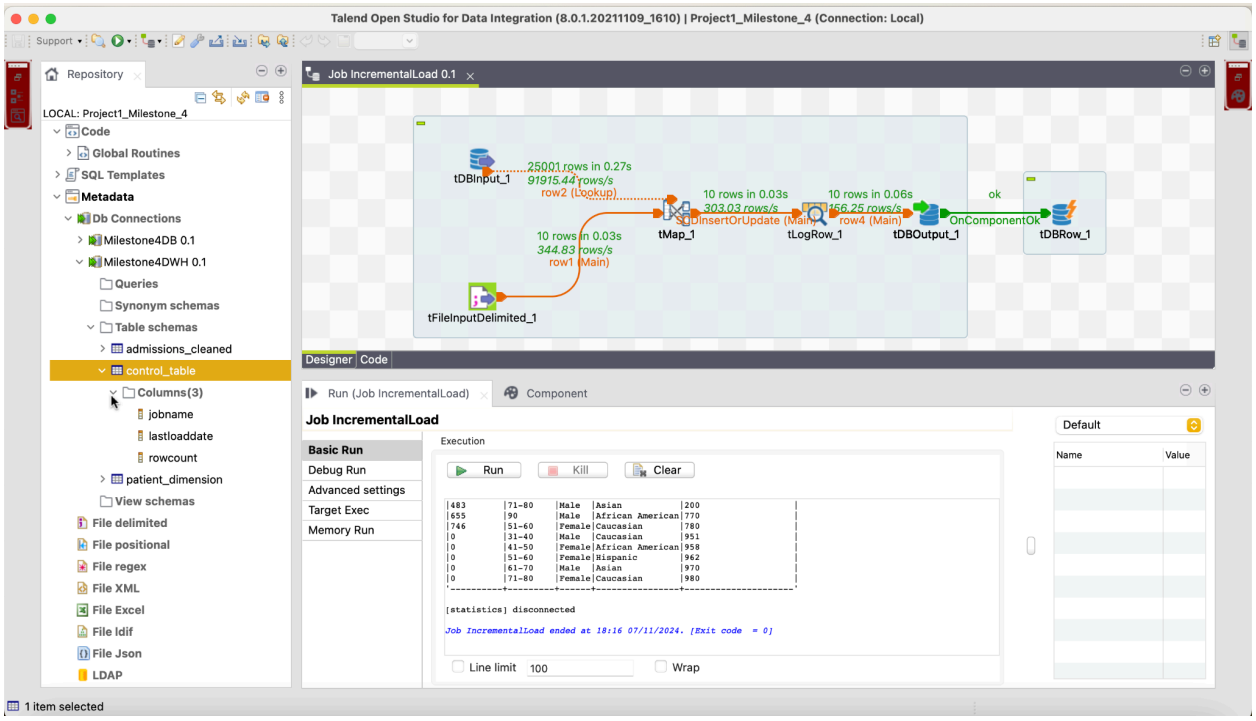
A dedicated control table documents each ETL execution, capturing metadata, ensuring accountability and enabling quick diagnostics. Managed by the wrapper job, this control table logs each ETL job in sequence, capturing essential details like job name, load date/time, and row count. The wrapper job orchestrates the ETL workflow, ensuring that jobs are executed in the correct order and updating the control table with metadata for each job as it completes. This setup provides a centralized record, offering visibility into the ETL workflow and enabling quick troubleshooting.

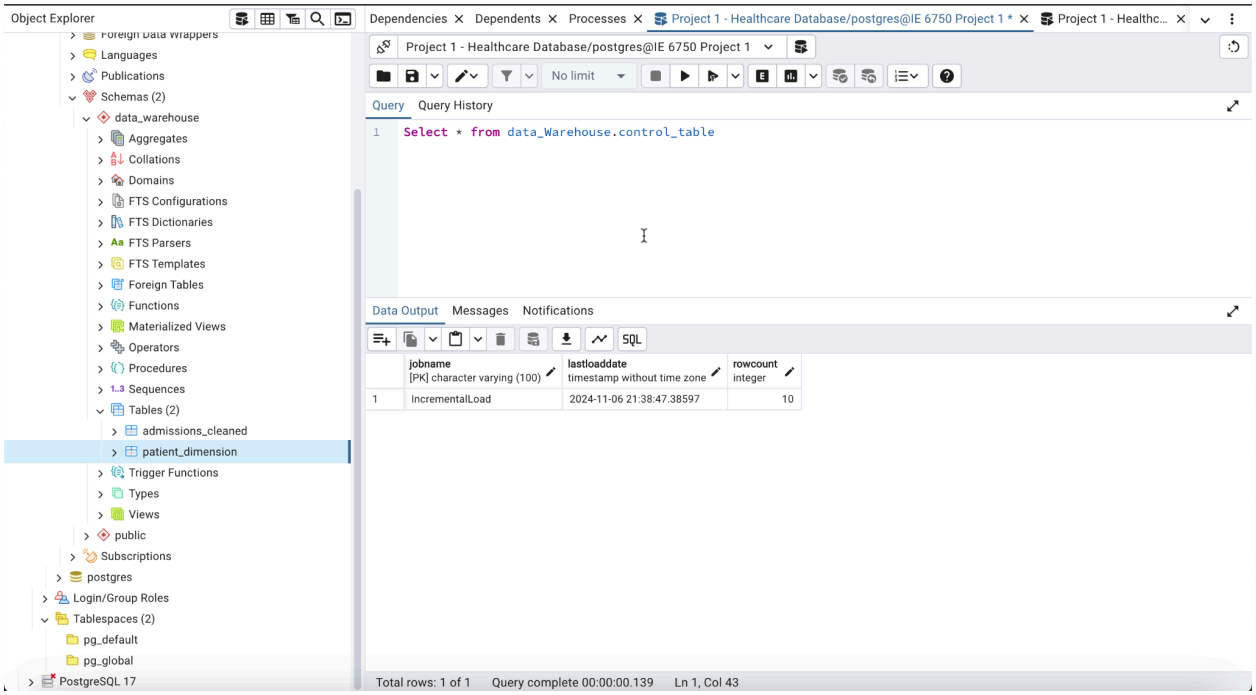
By documenting when and how many rows are processed for each job, the control table ensures incremental loads occur as expected and that the data warehouse remains up-to-date. This table is invaluable for monitoring and auditing, allowing the data engineering team to verify job execution, maintain data integrity across the pipeline, and support both data governance and compliance efforts.

In the future, this control table will remain an essential resource for operational monitoring and auditing. With a clear log of each ETL job and the sequence managed by the wrapper job, the data engineering team can quickly verify if jobs are running as scheduled and identify any points of failure. For instance, if a load fails or an error is recorded, the team can rely on this table to assess and address the issue without manual guesswork. Additionally, having a timestamped log of all ETL activities managed through the wrapper job supports data governance and compliance efforts, ensuring that data integrity is maintained consistently throughout the pipeline.

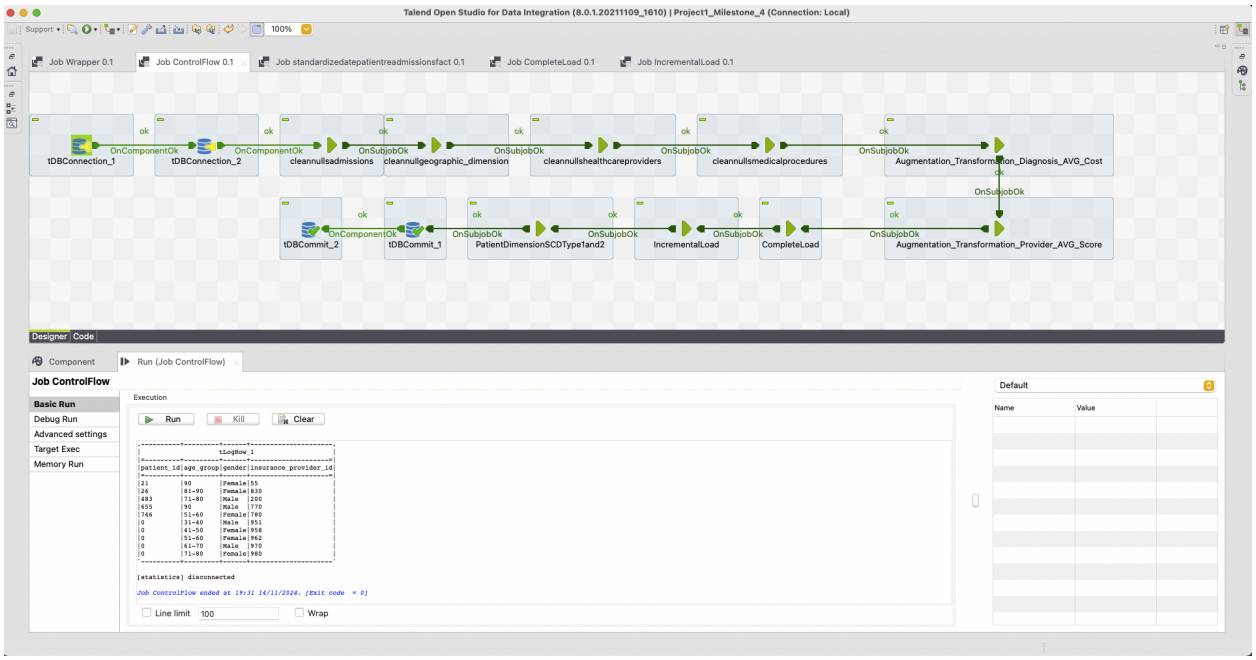


# Final Project 1 Report



# Final Project 1 Report




## Conclusion

In this project, we developed a comprehensive data warehouse for tracking hospital readmissions, using PostgreSQL for data storage and Talend for designing and managing the ETL workflows. From initial data sourcing to implementing both complete and incremental loads, we worked to ensure that our data warehouse remains accurate, current, and valuable for analyzing readmission trends.

Throughout the project, we encountered several challenges. One of the key issues was configuring incremental loads accurately to capture only new or modified records without duplicating data. We addressed this by carefully implementing time-stamped filters in Talend, allowing us to update our warehouse efficiently without unnecessary data processing. Another challenge was ensuring accurate row counts and tracking in our control table to maintain transparency across ETL jobs. By adding detailed logging and error-handling steps, we were able to monitor job status effectively and quickly troubleshoot any inconsistencies.

Using Talend, we configured ETL workflows with necessary transformations, aggregations, and control tracking, logging each job's details—such as load timestamps and row counts—to ensure accountability and quick diagnostics. Aggregated metrics, including average costs by diagnosis and satisfaction levels by provider, offer healthcare administrators actionable insights into factors impacting readmissions and resource allocation.

We aimed to demonstrate the potential of a well-structured data warehouse to support data-driven decision-making in healthcare. Future enhancements could involve integrating additional data sources, optimizing ETL performance, and exploring predictive analytics to

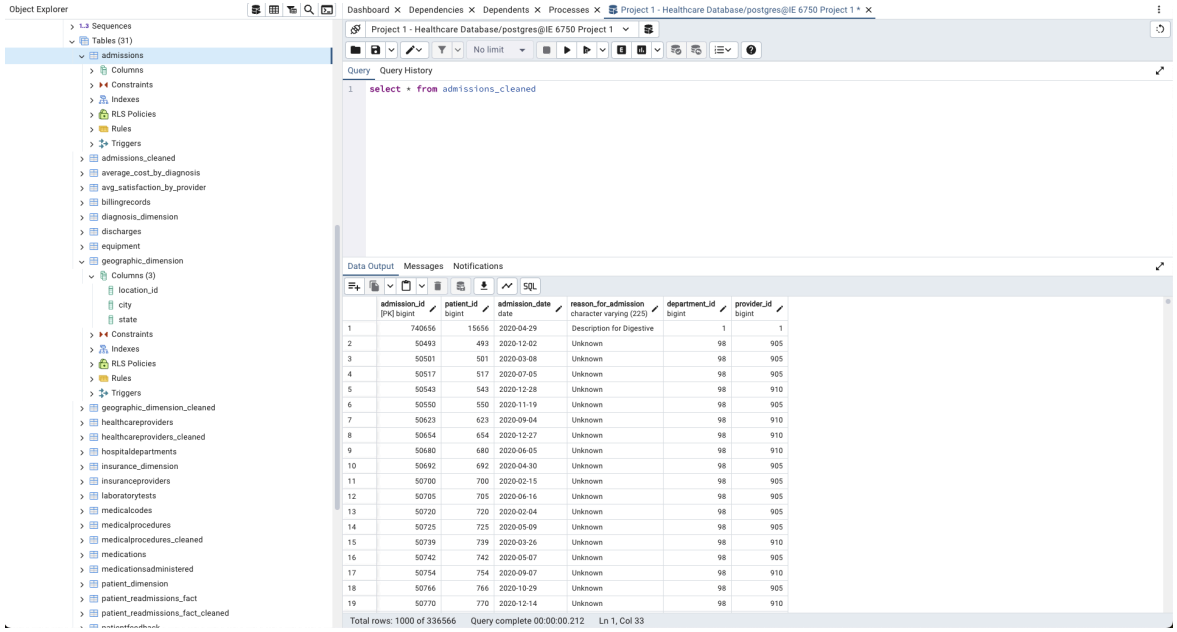
# Final Project 1 Report

further support patient care and operational efficiency. This experience provided us with valuable hands-on knowledge in data warehousing, ETL workflows, and healthcare analytics, skills that will be instrumental in our professional and academic development.

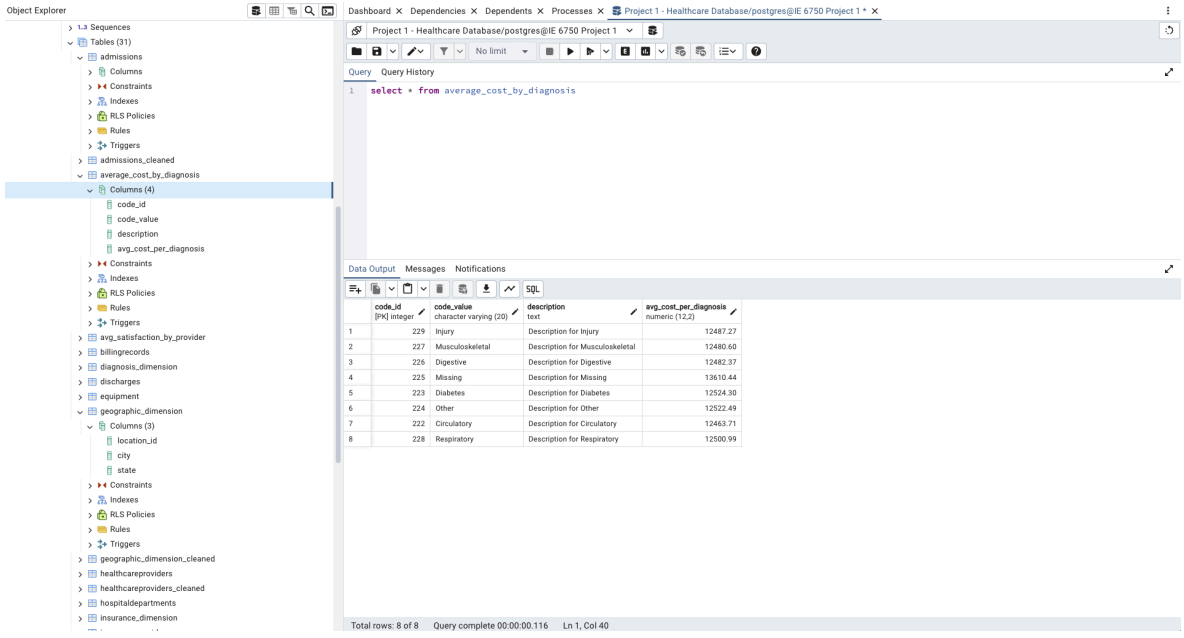
## Appendix:

### i. Postgres Implementation

- **Admissions**

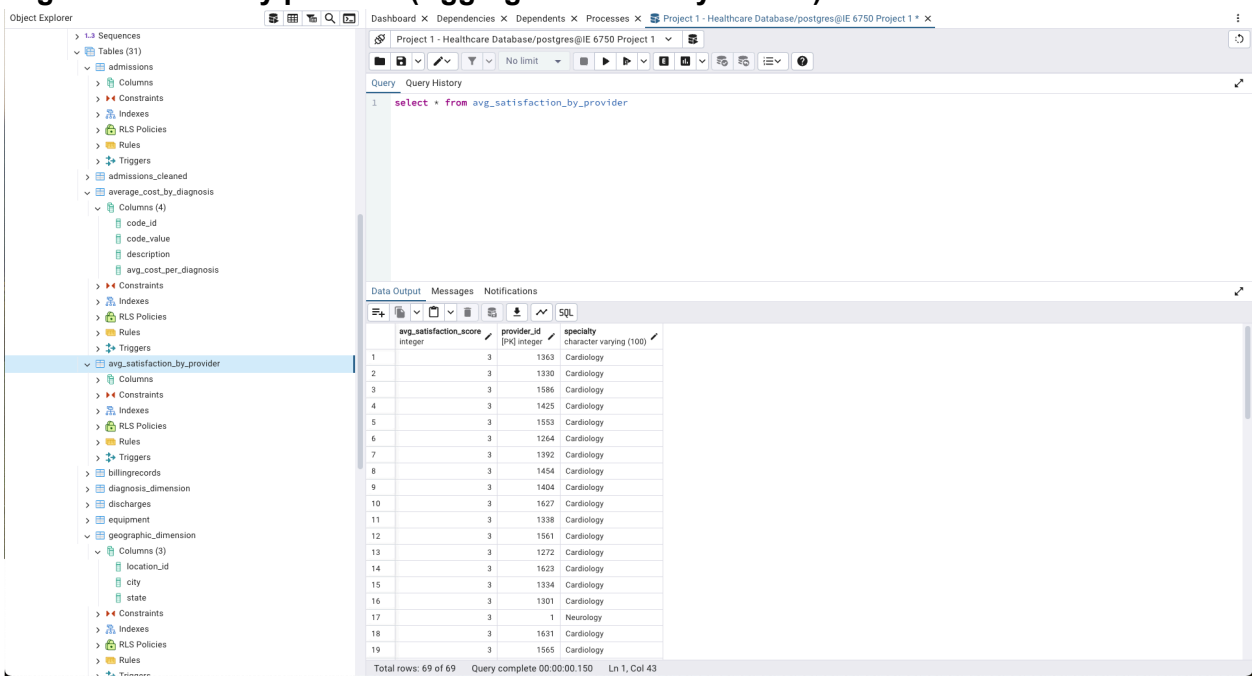



- **Avg_cost_by_diagnosis (Aggregation performed by Talend)**

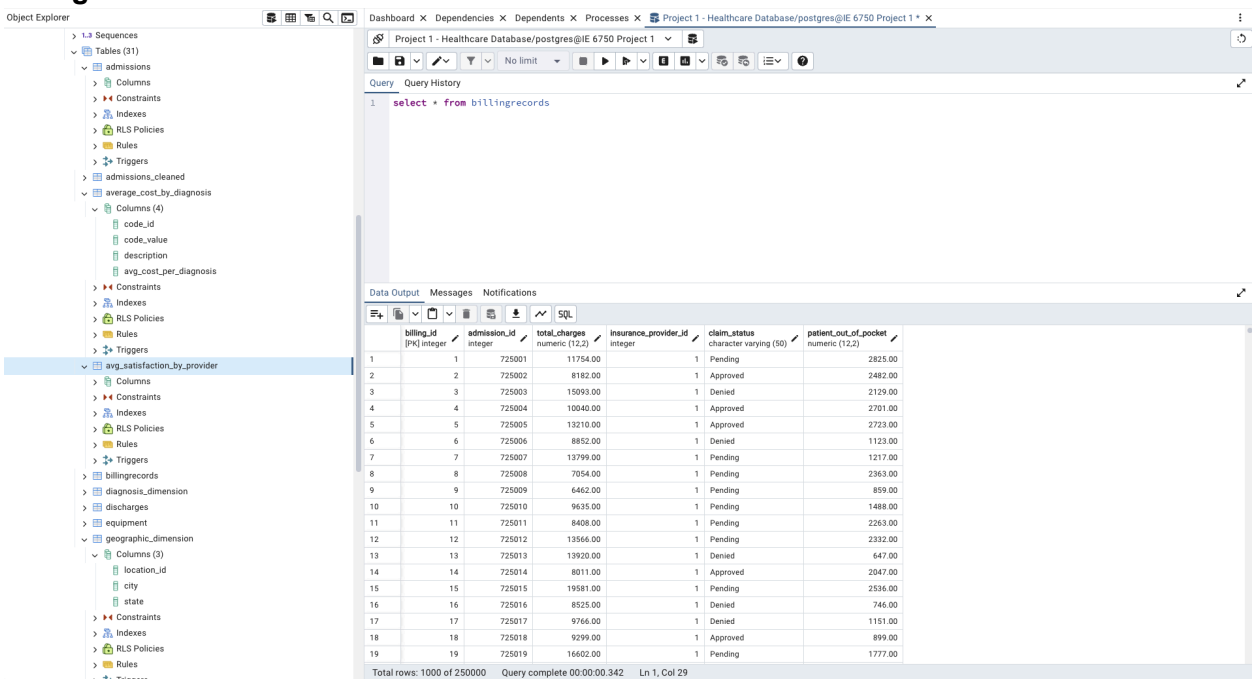


# Final Project 1 Report

- **Avg. satisfaction by provider (Aggregation done by Talend)**

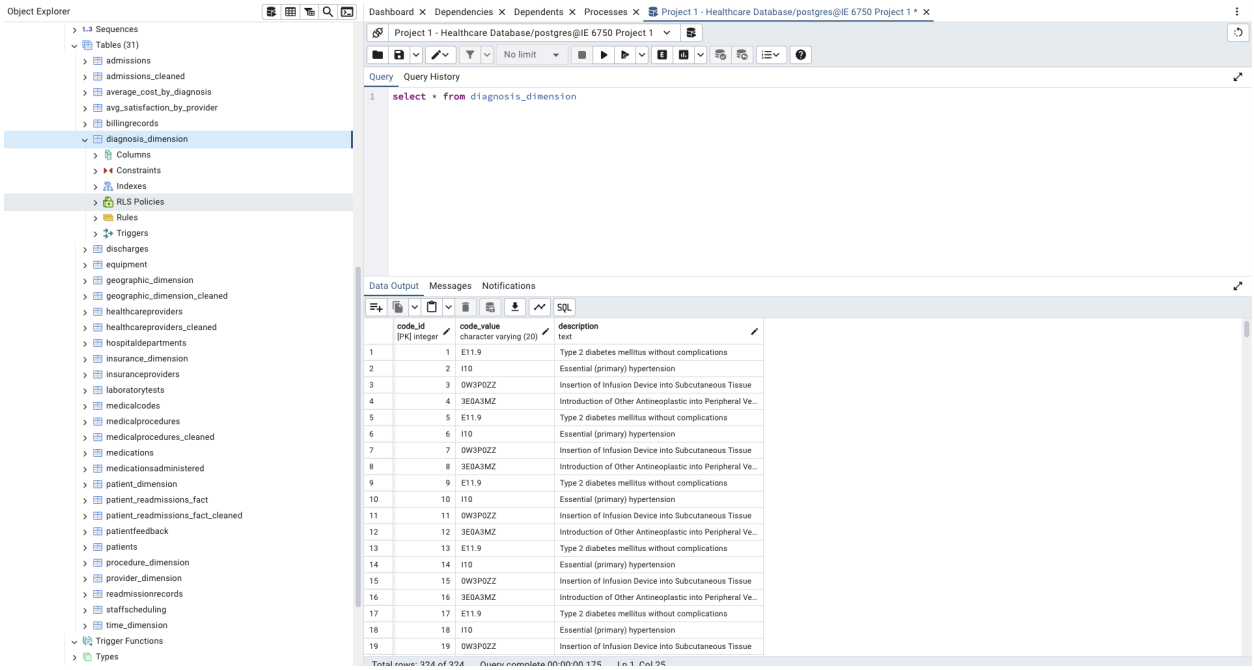



- **Billing Records**

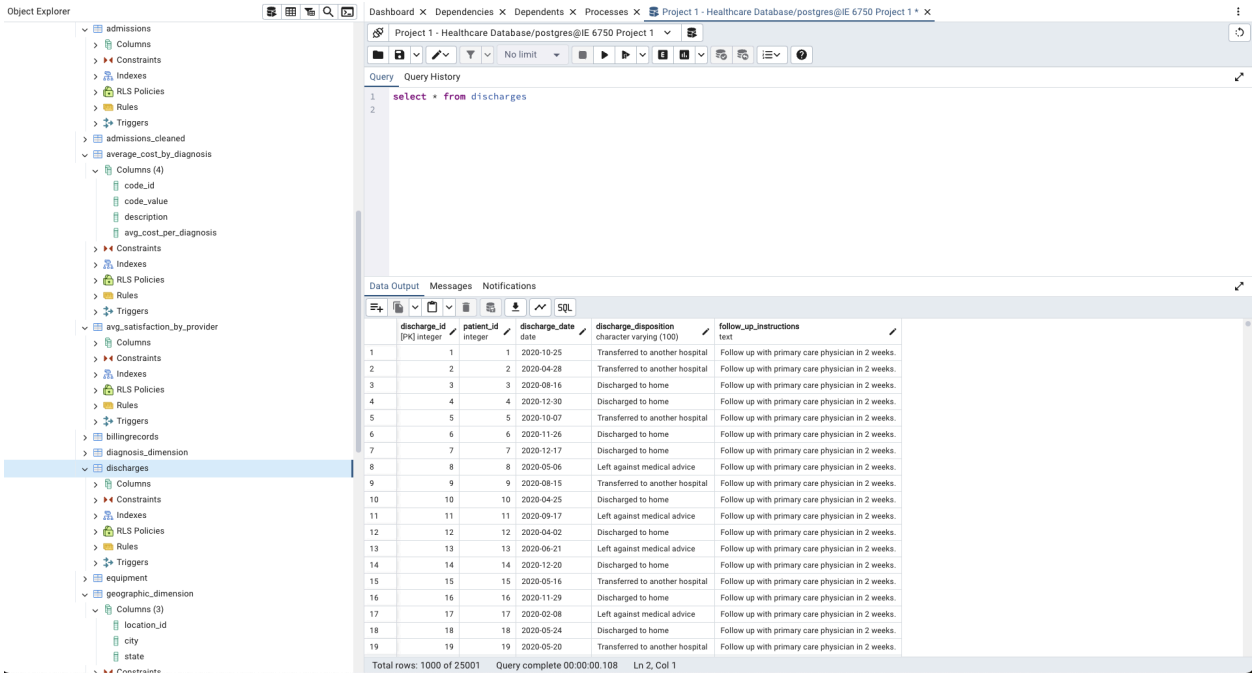


# Final Project 1 Report

- ## Diagnosis dimension

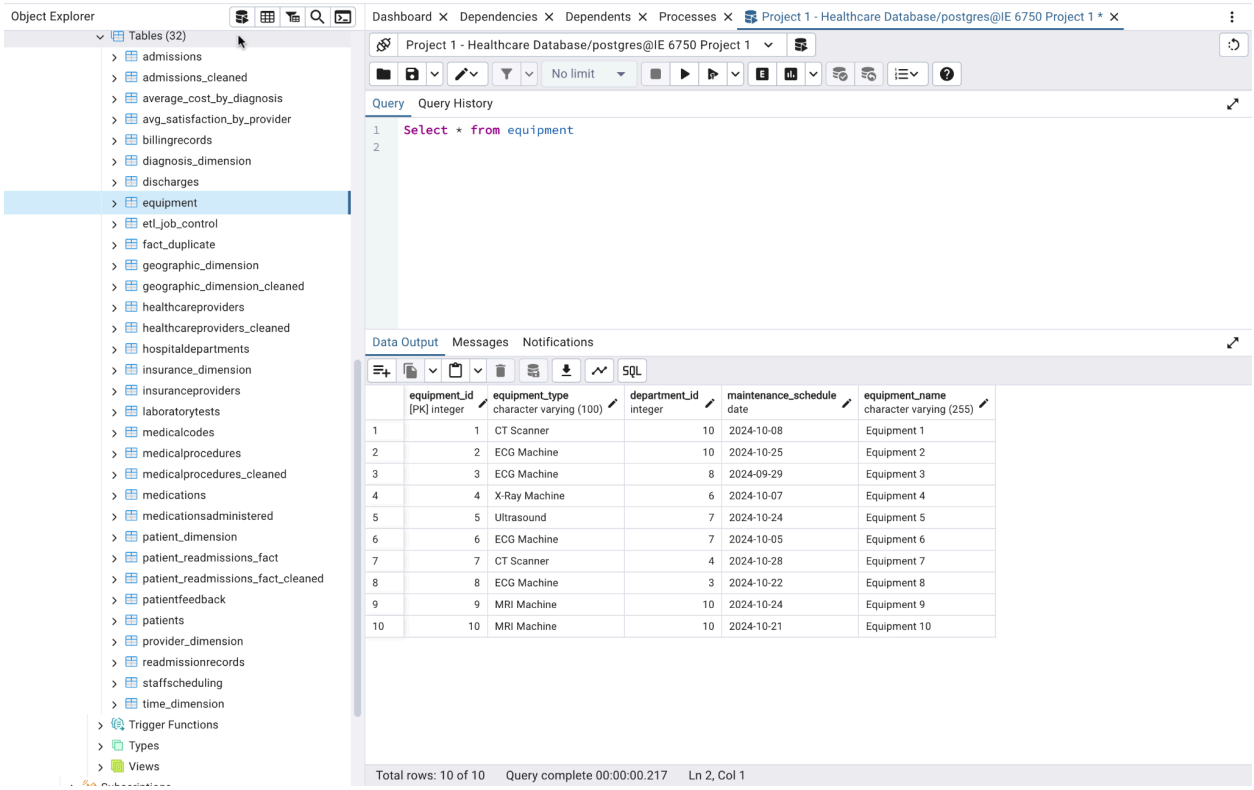



- ## Discharges

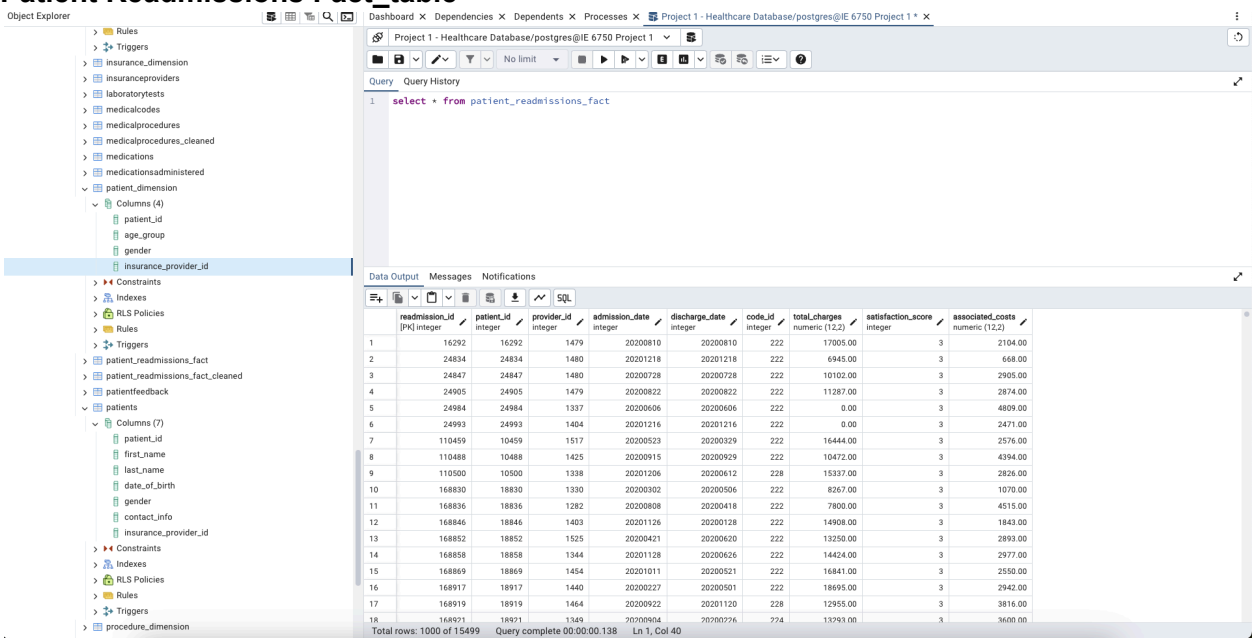


# Final Project 1 Report

- **Equipment**

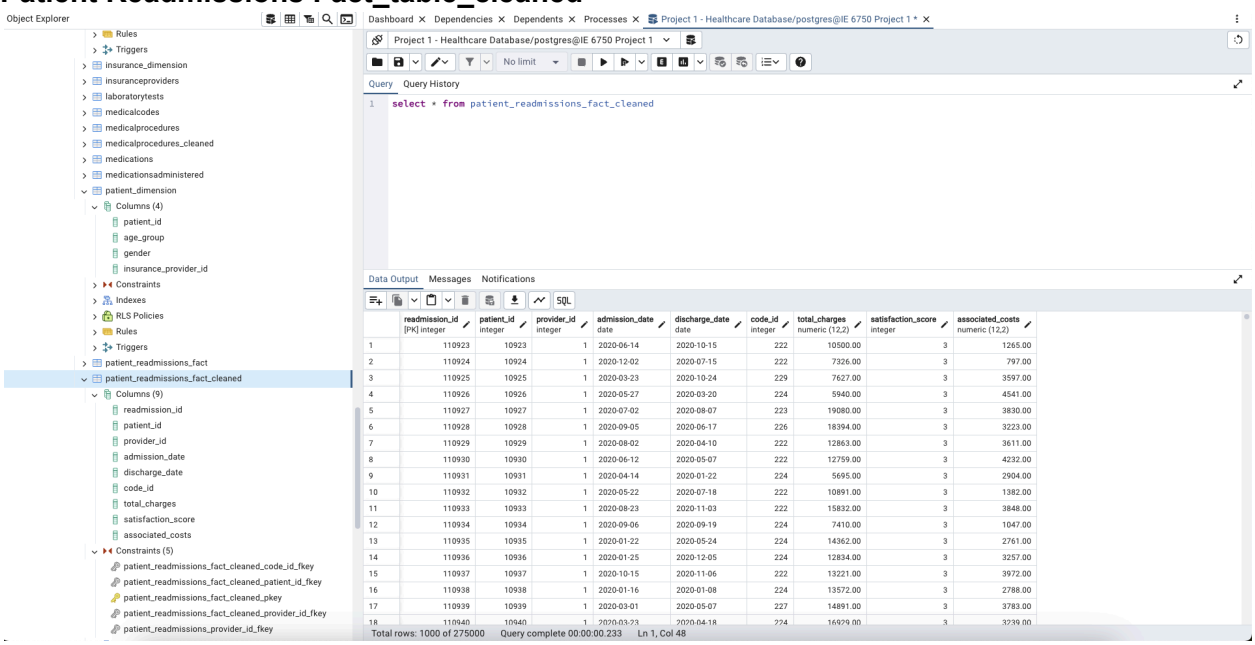



- **Patient Readmissions Fact_table**

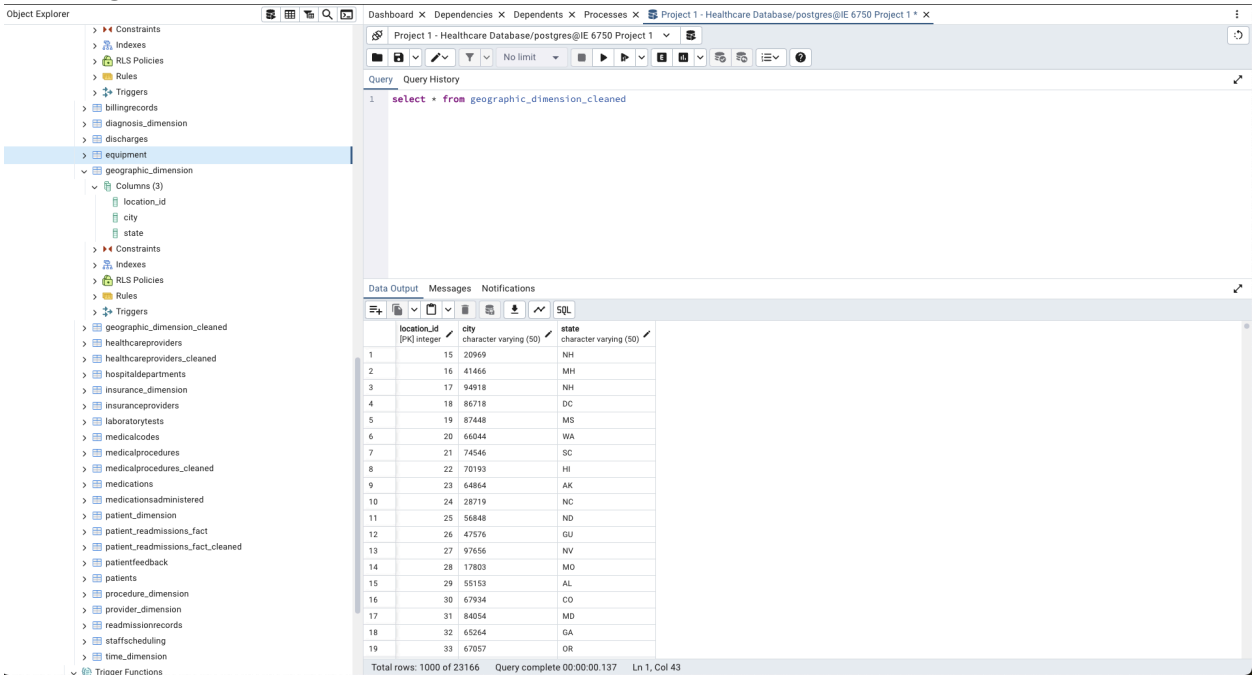


# Final Project 1 Report

- **Patient Readmissions Fact_table_cleaned**

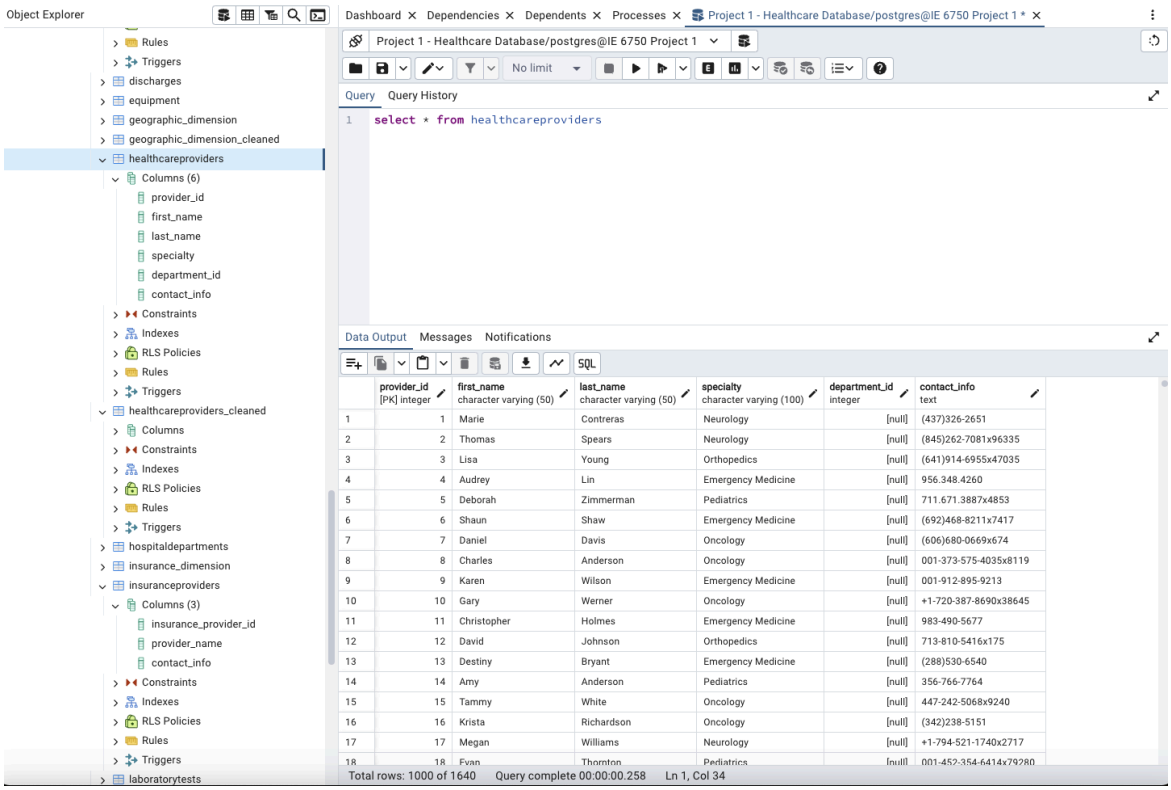



## 10. Geographic dimension Cleaned

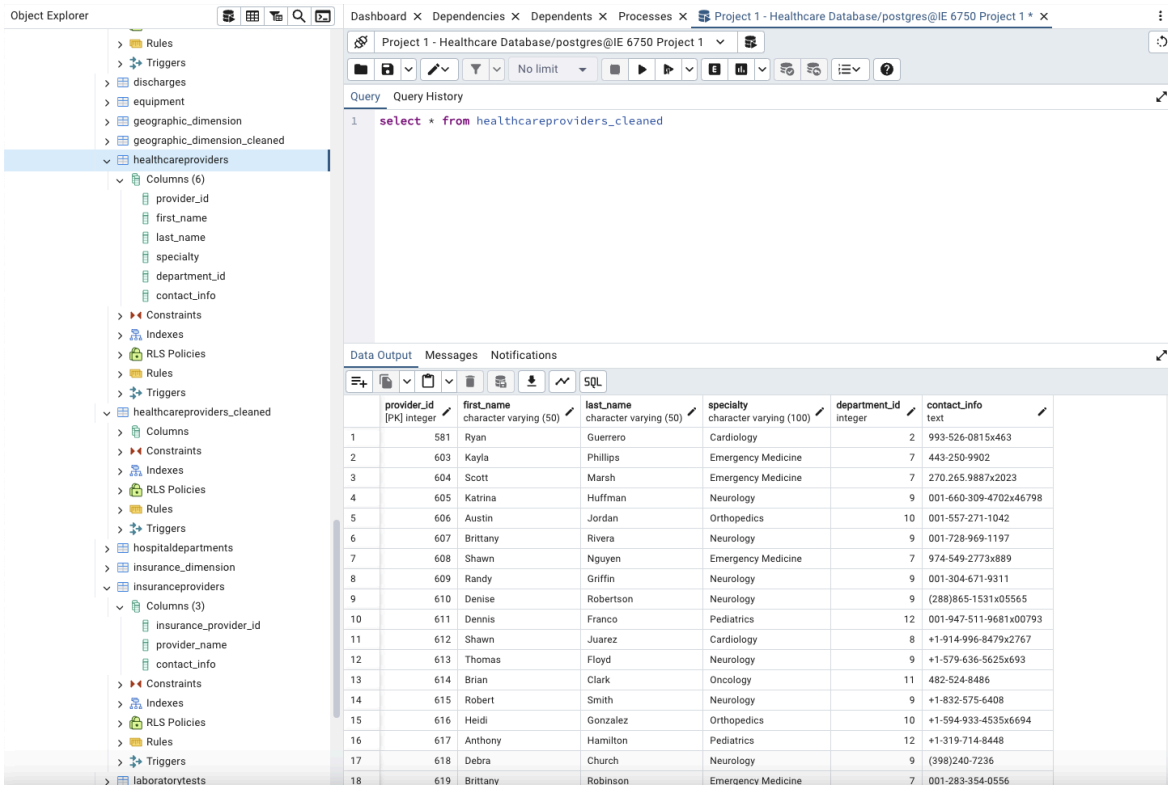


# Final Project 1 Report

## 11. Healthcare Providers

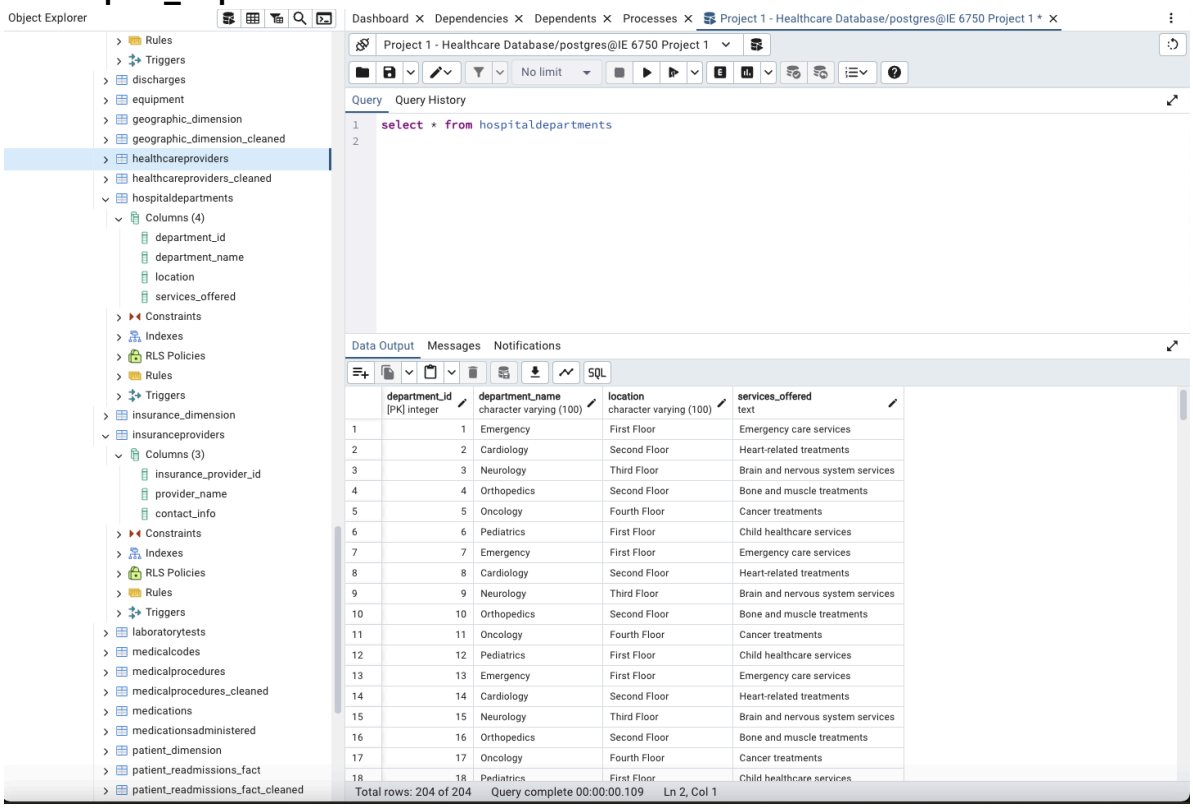



## 12. Healthcare Providers_cleaned

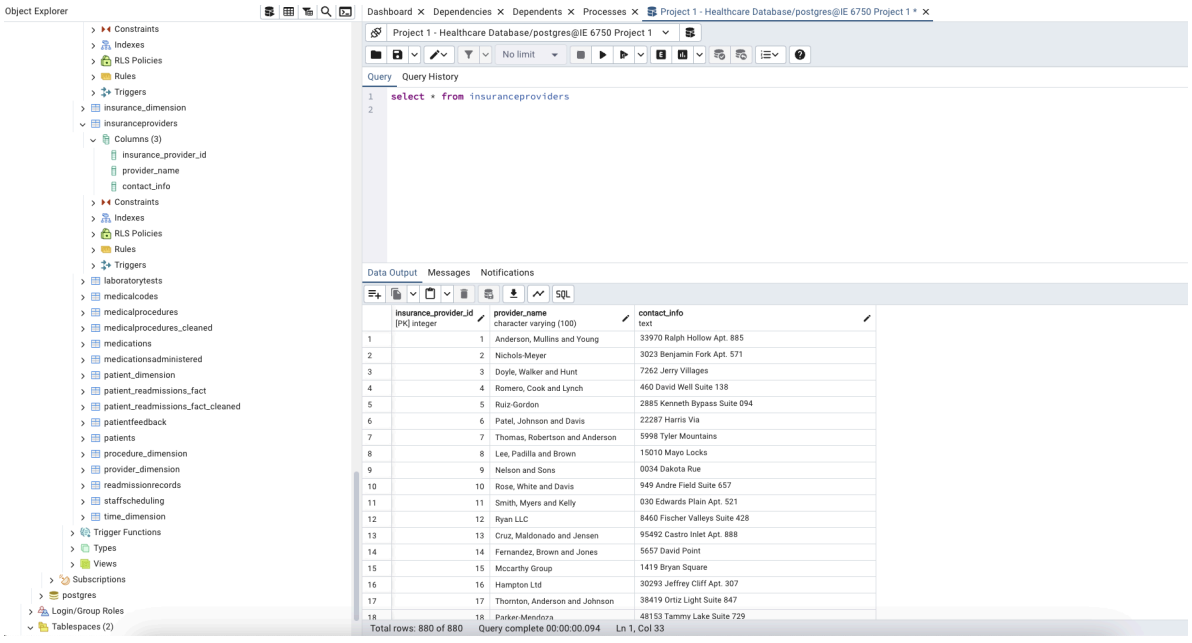


# Final Project 1 Report

## 13. Hospital_Departments

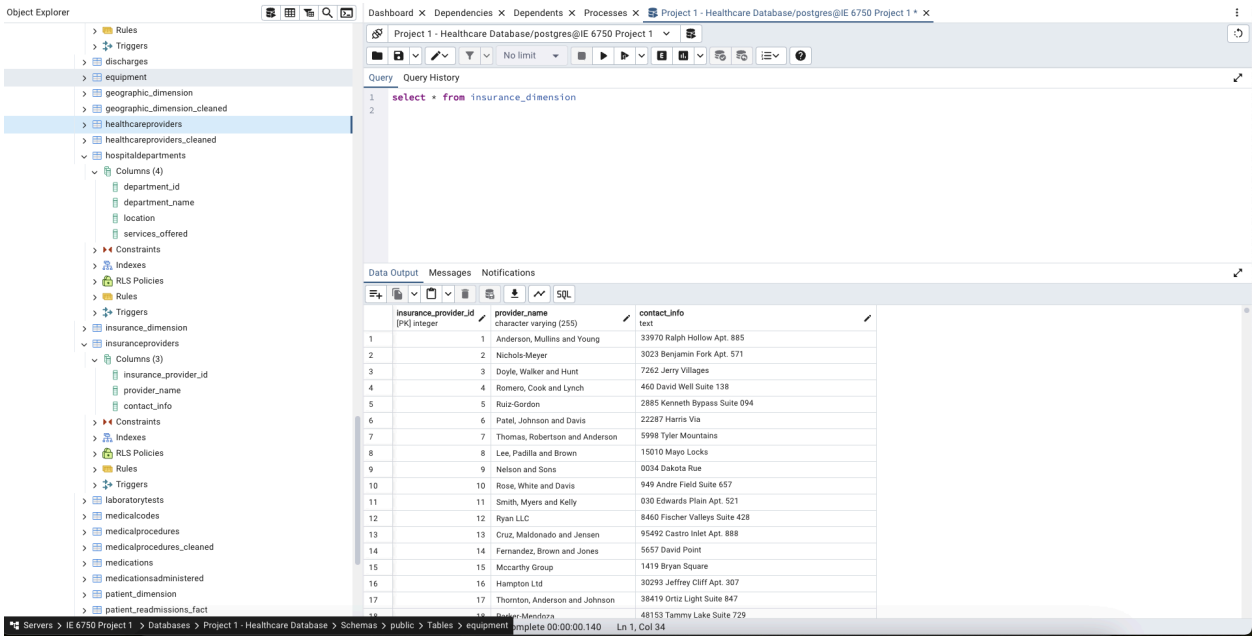



## 14. Insurance Providers

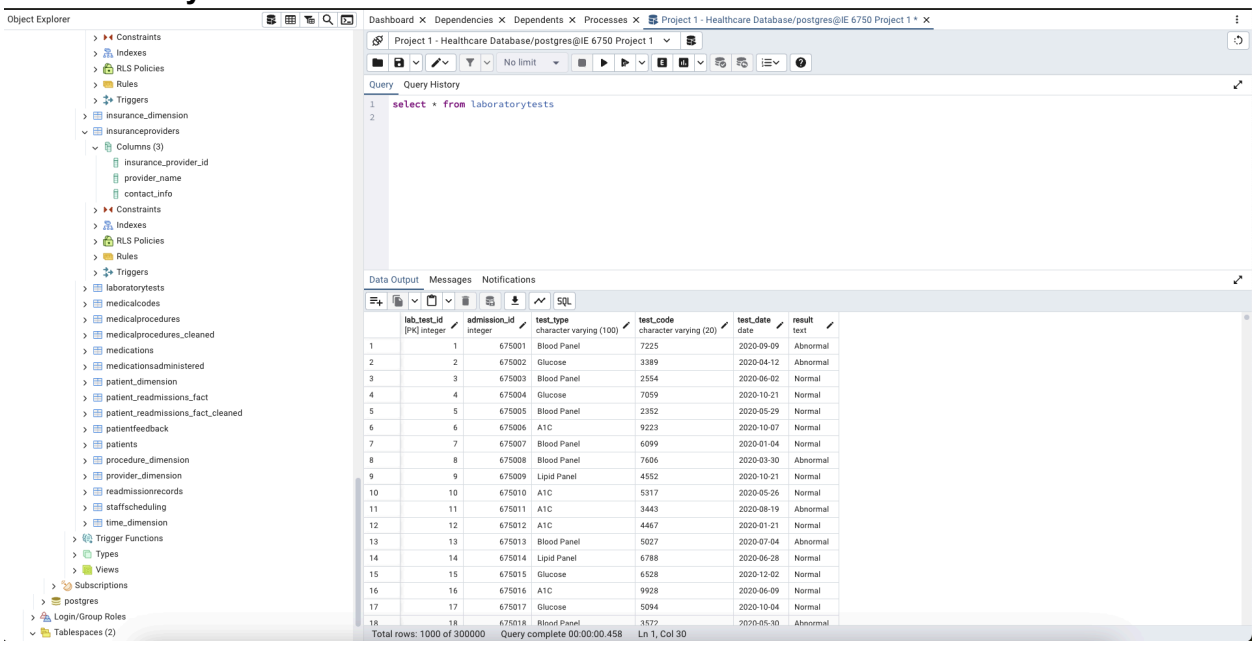


# Final Project 1 Report

## 15. Insurance dimension

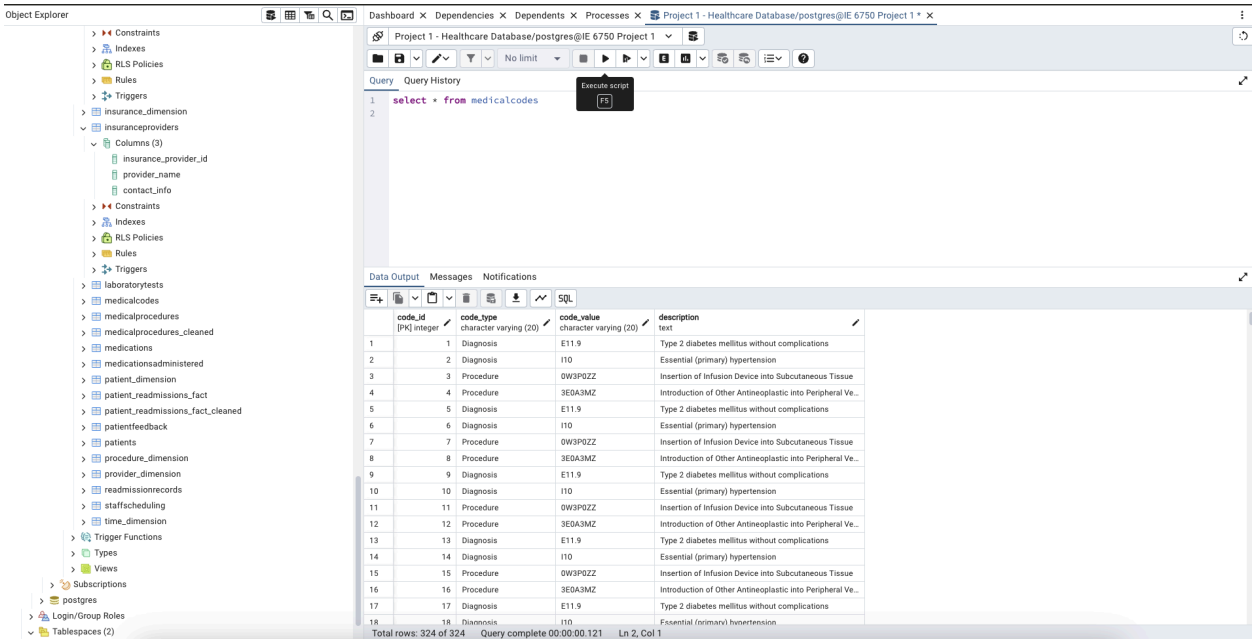



## 16. Laboratory Tests

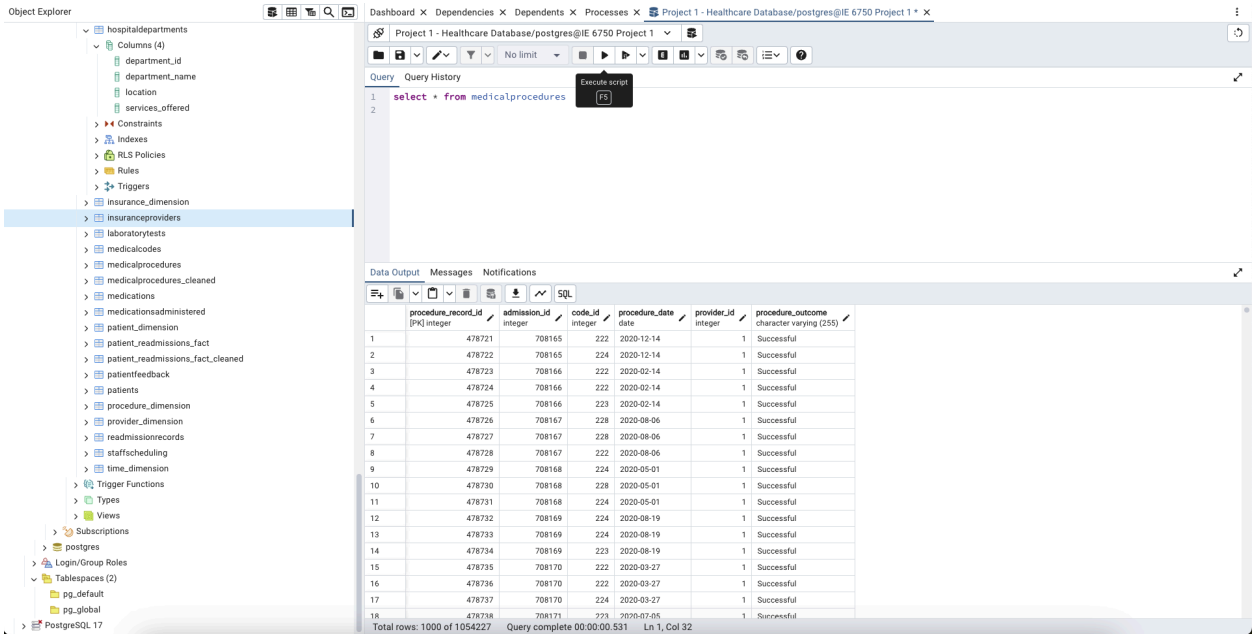


# Final Project 1 Report

## 17. Medical Codes

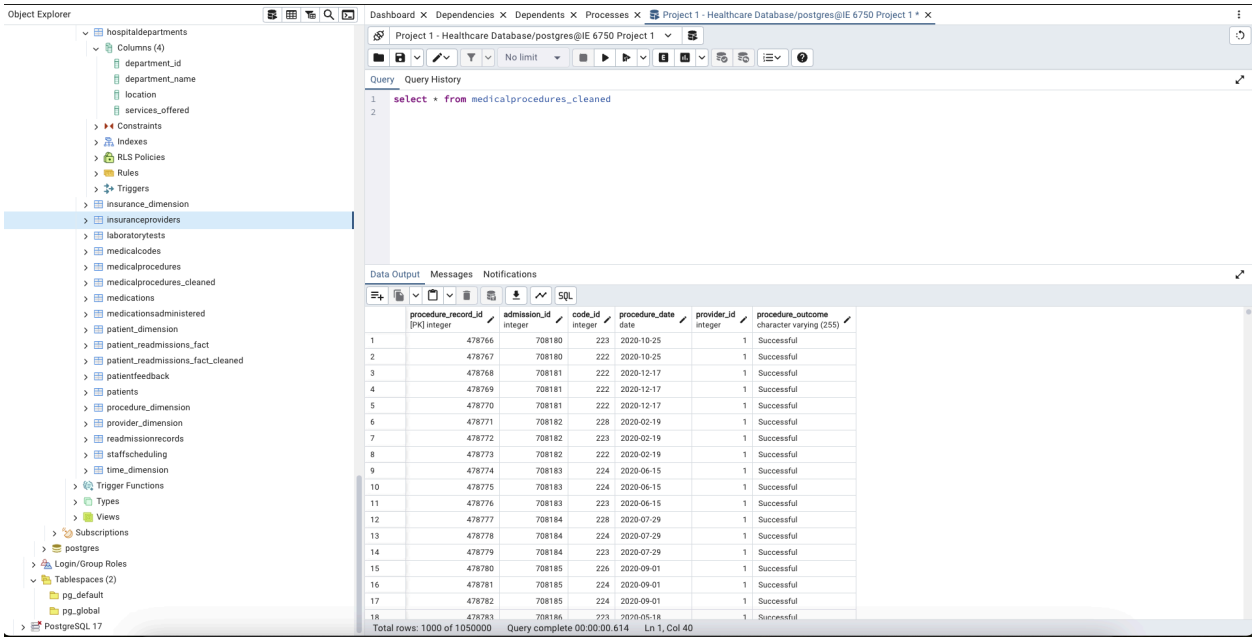



## 18. Medical Procedures

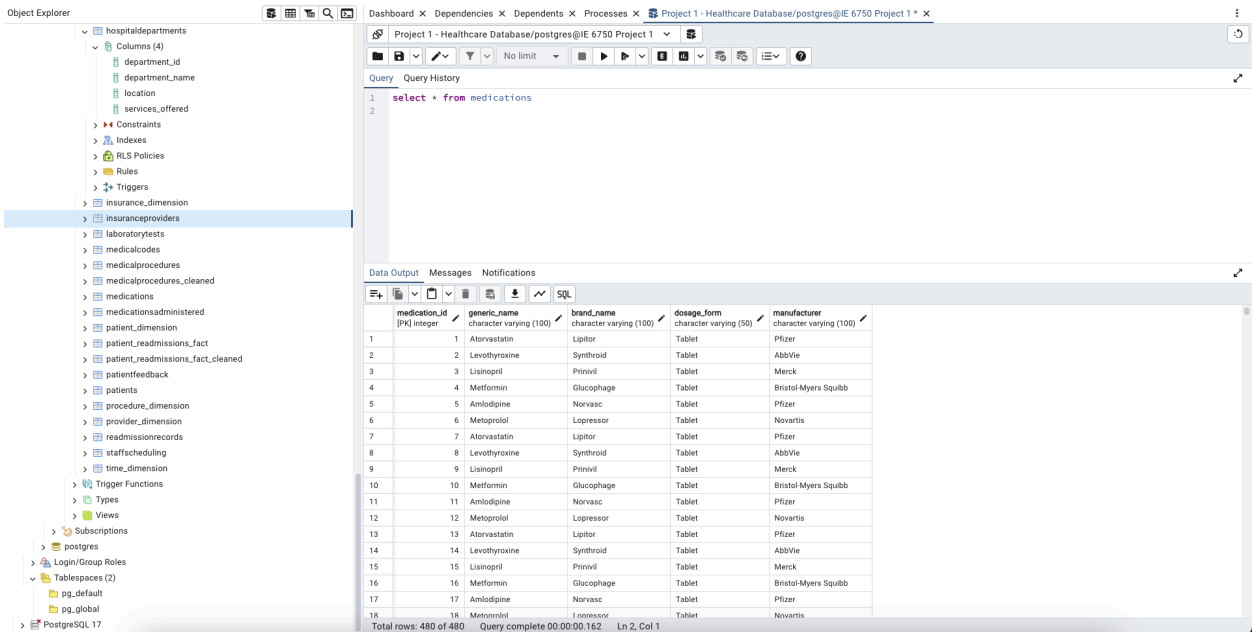


# Final Project 1 Report

## 19. Medical Procedures Cleaned

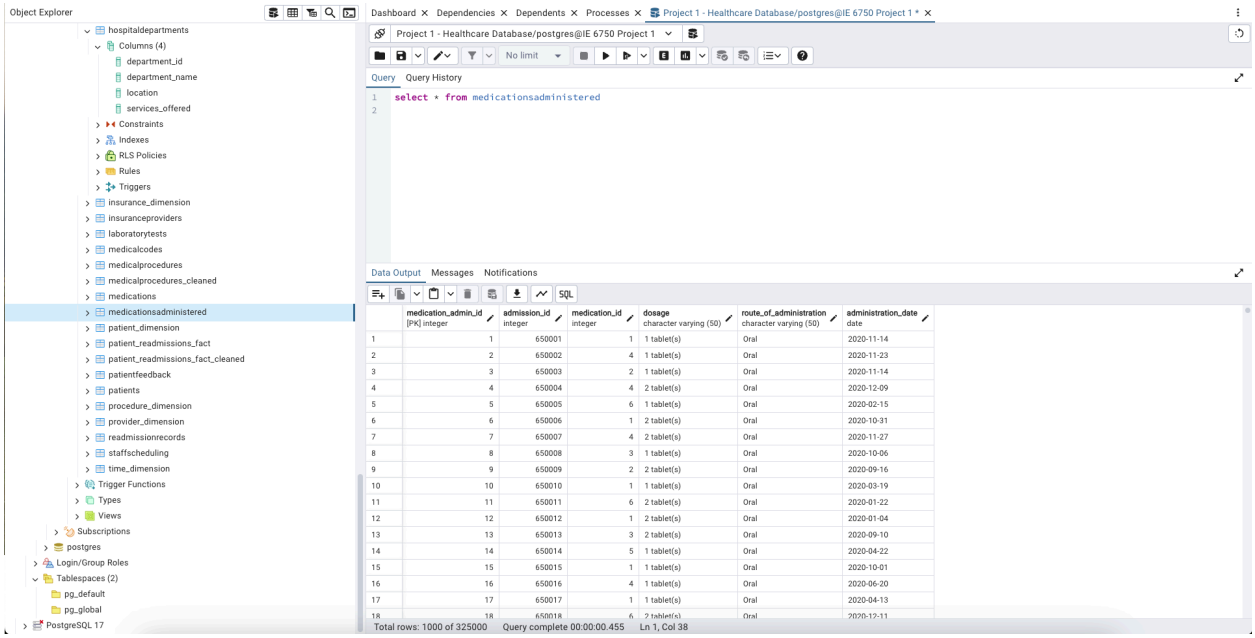



## 20. Medications

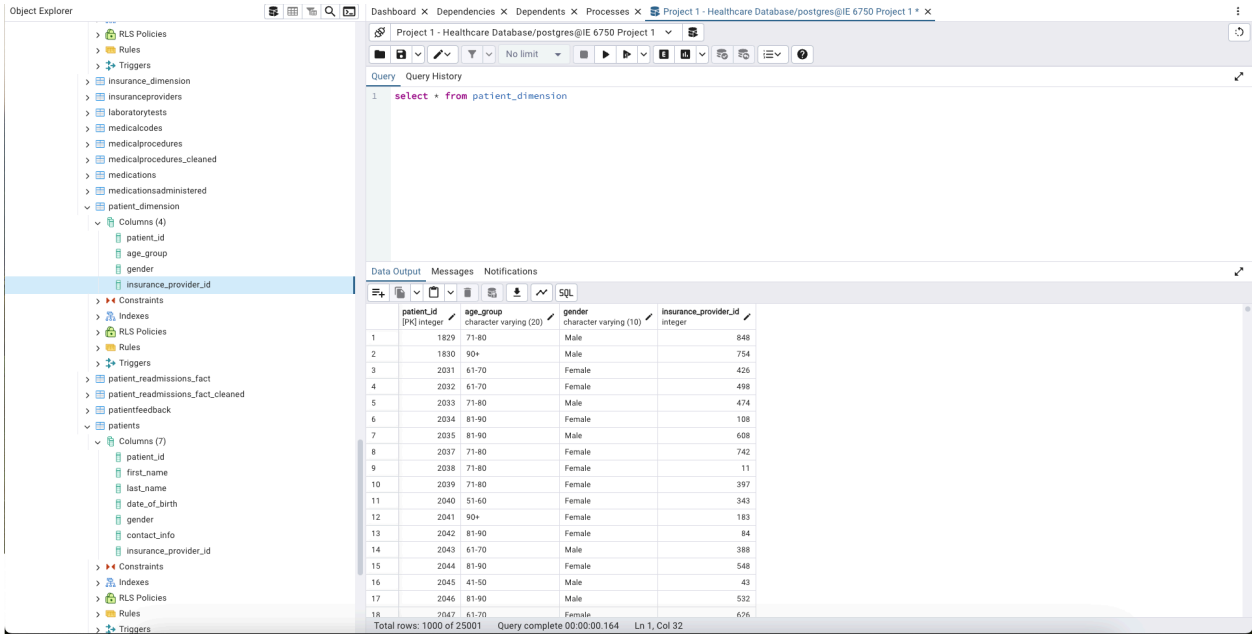


# Final Project 1 Report

## 21. Medications Administered

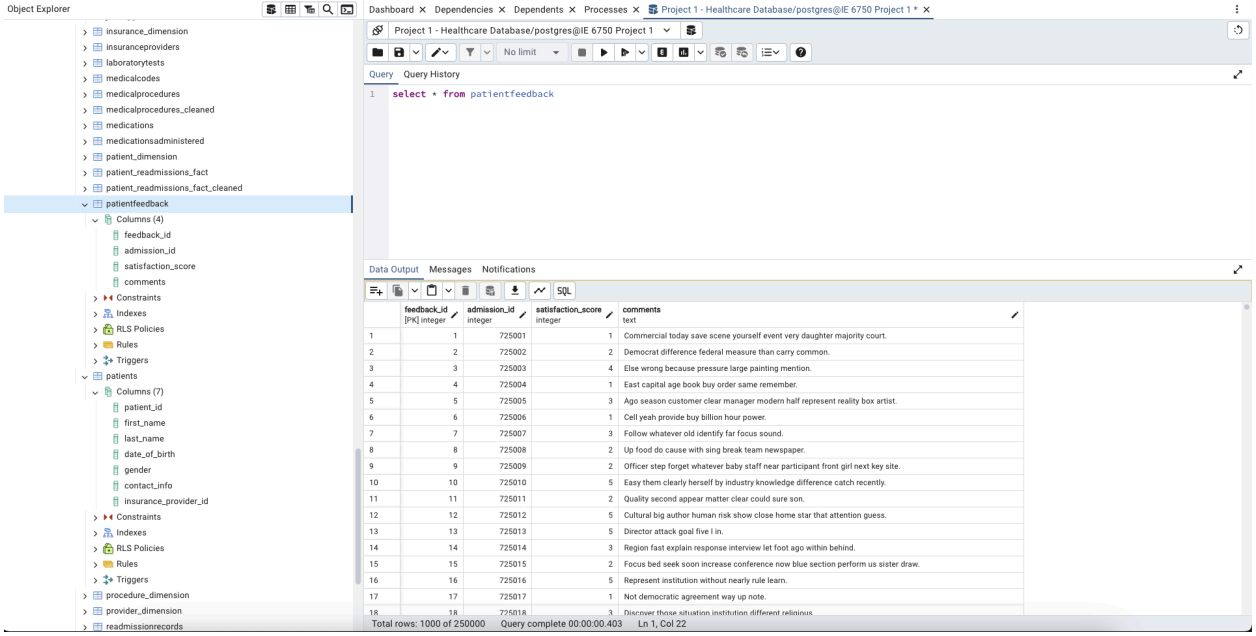



## 22. Patient dimension

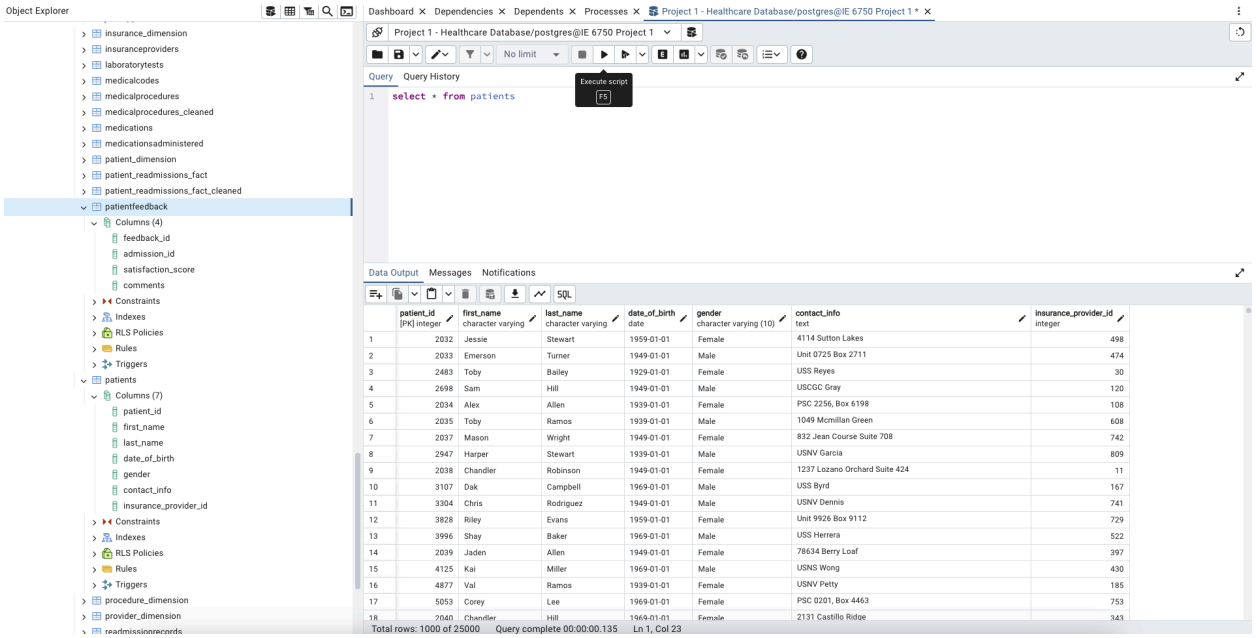


# Final Project 1 Report

## 23. Patient Feedback

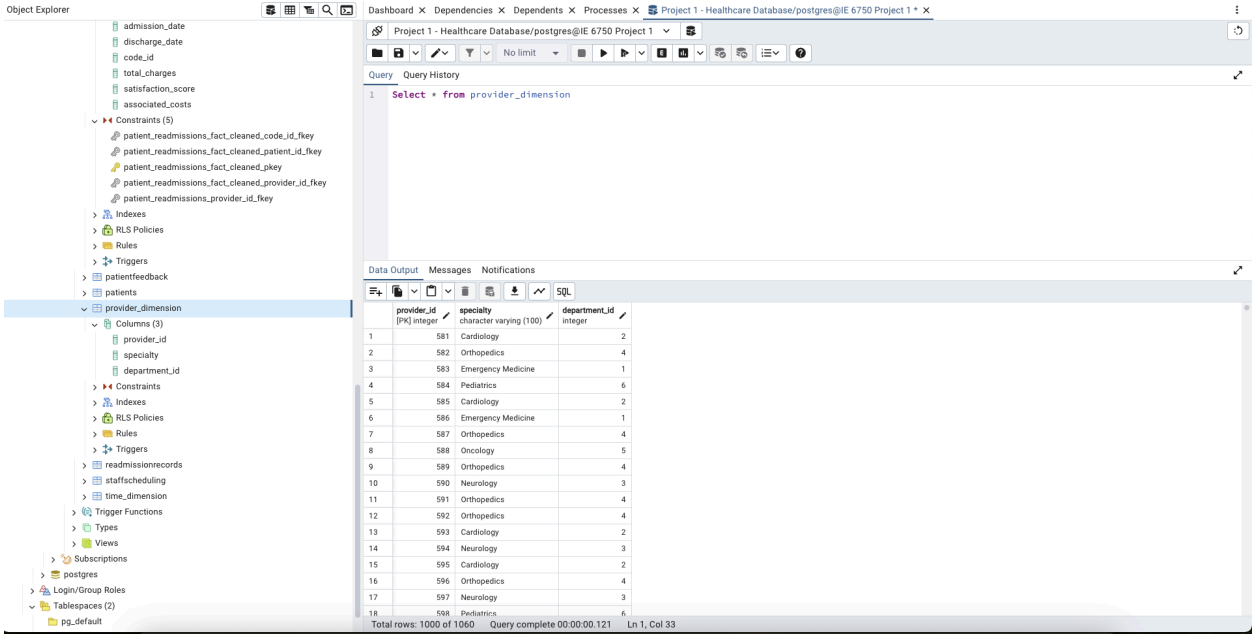



## 24. Patients

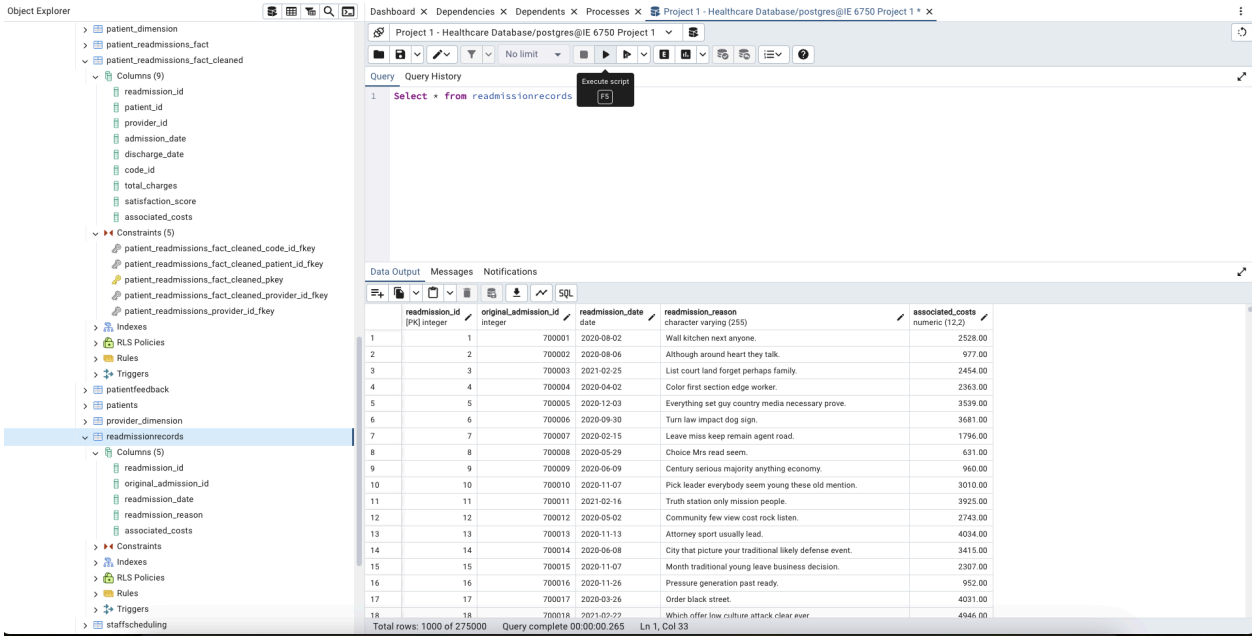


# Final Project 1 Report

## 25. Provider dimension

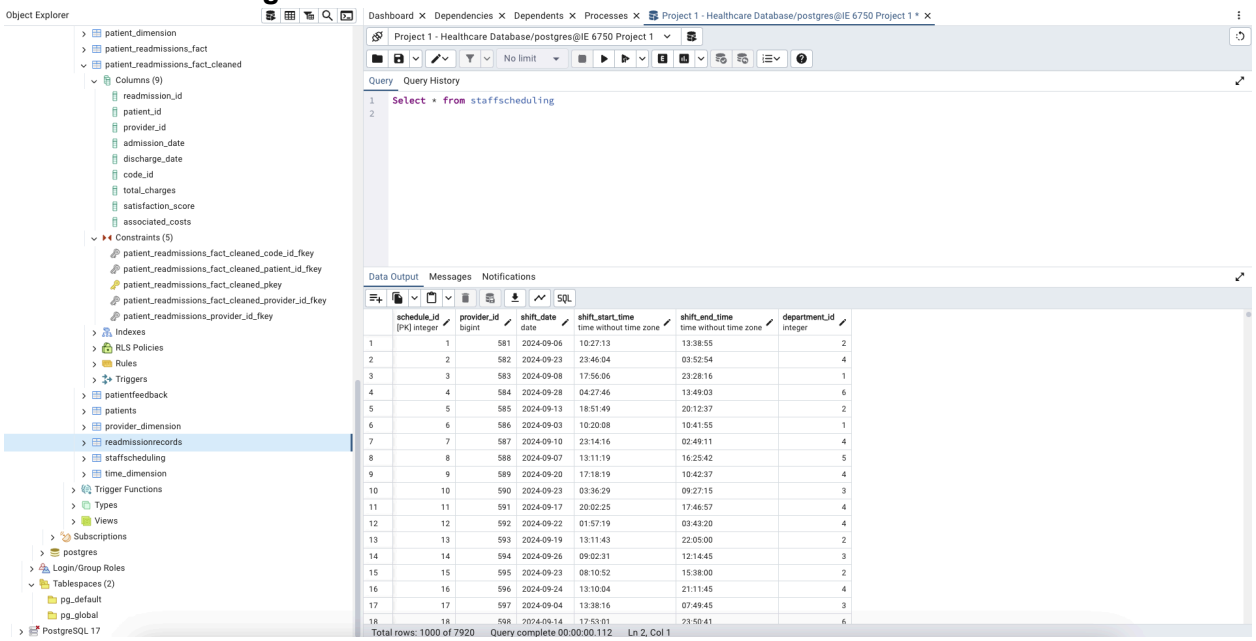



## 26. Readmission Records

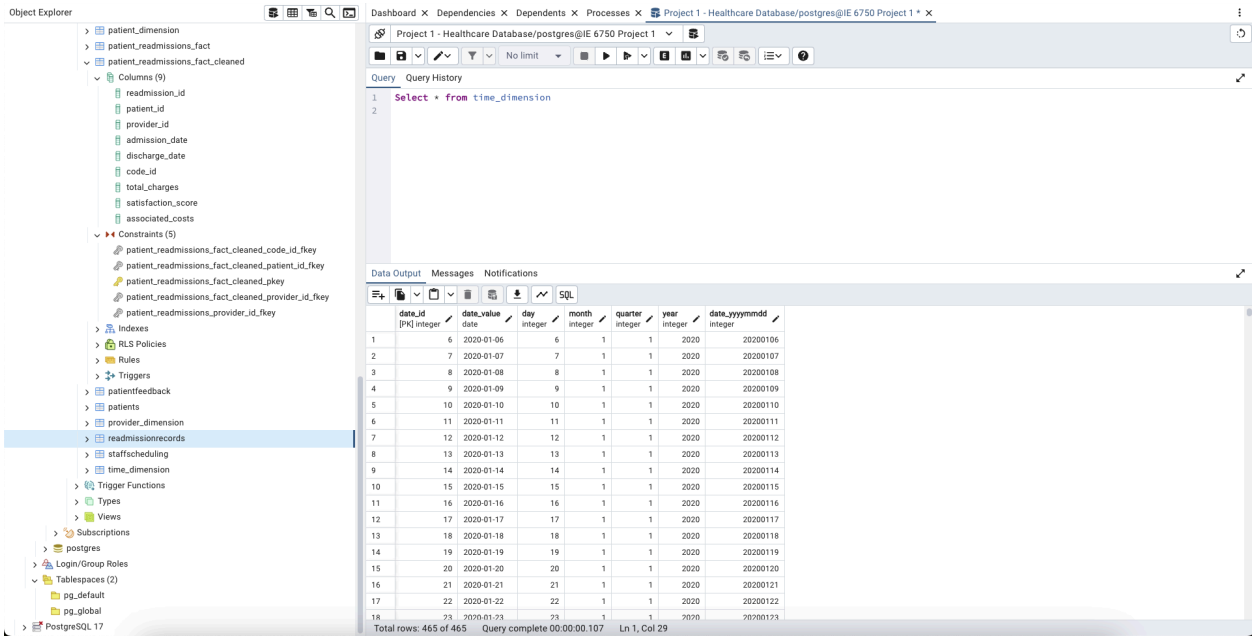


# Final Project 1 Report

## 27. Staff Scheduling



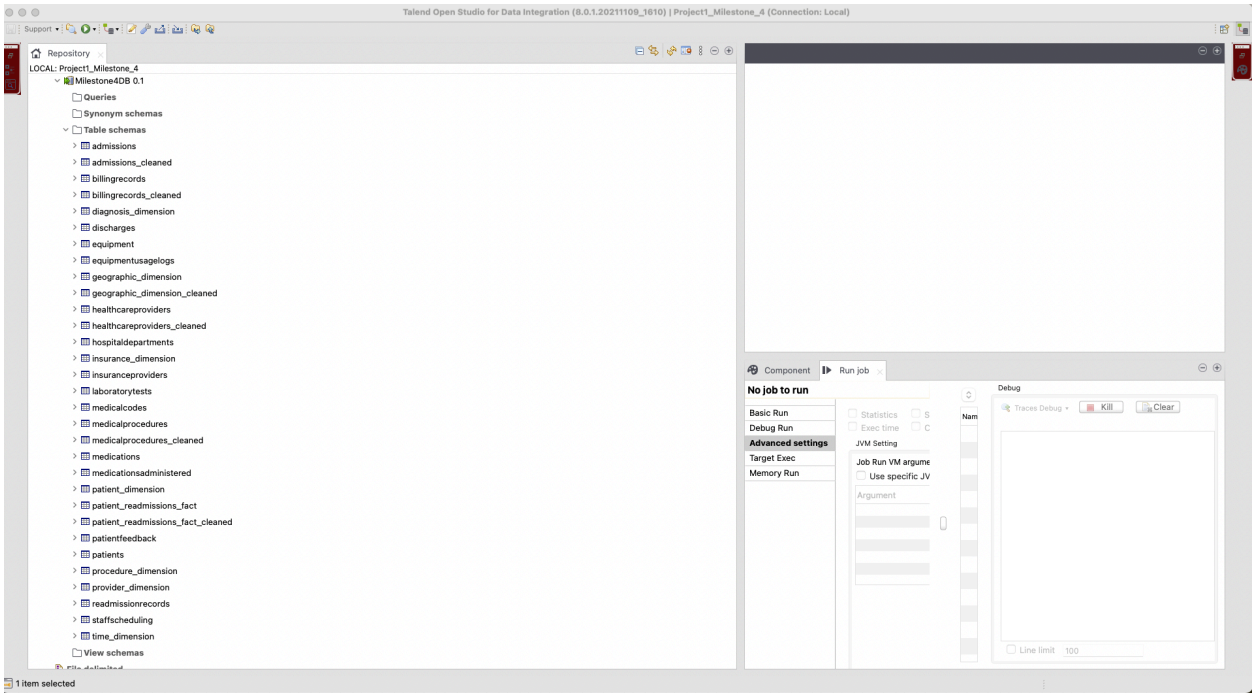

## 28. Time dimension



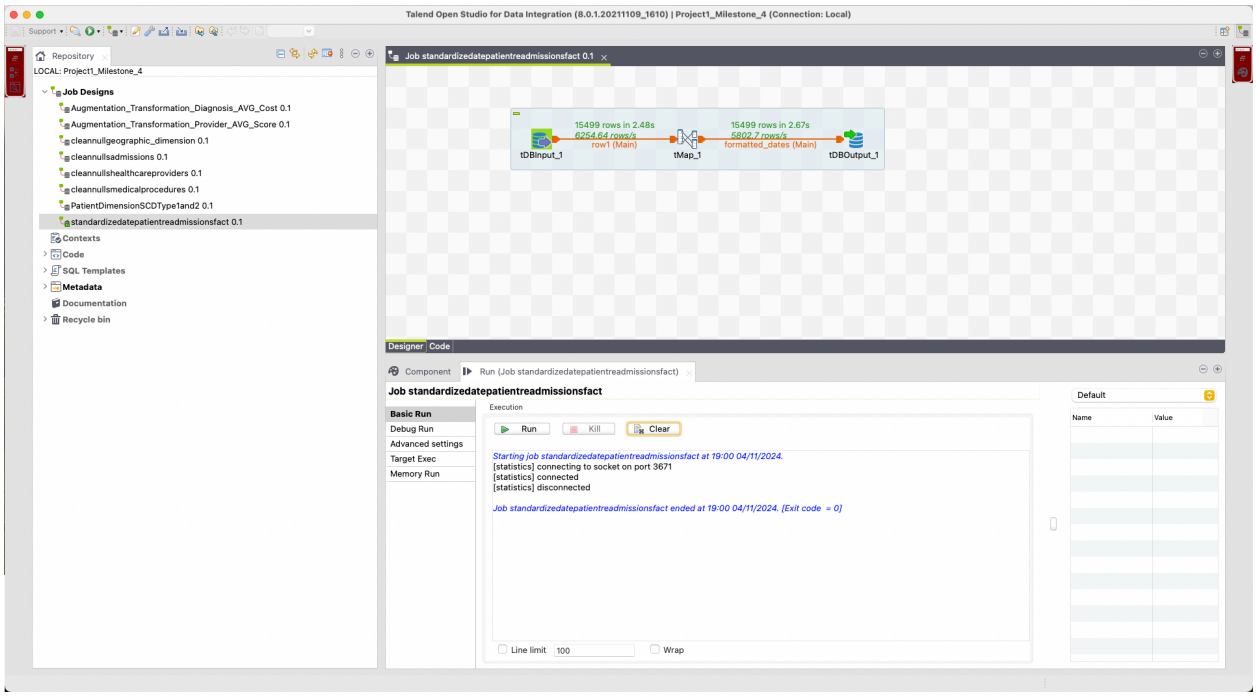
# Final Project 1 Report

## ii. Talend Implementation

- **Loaded Tables from Database**

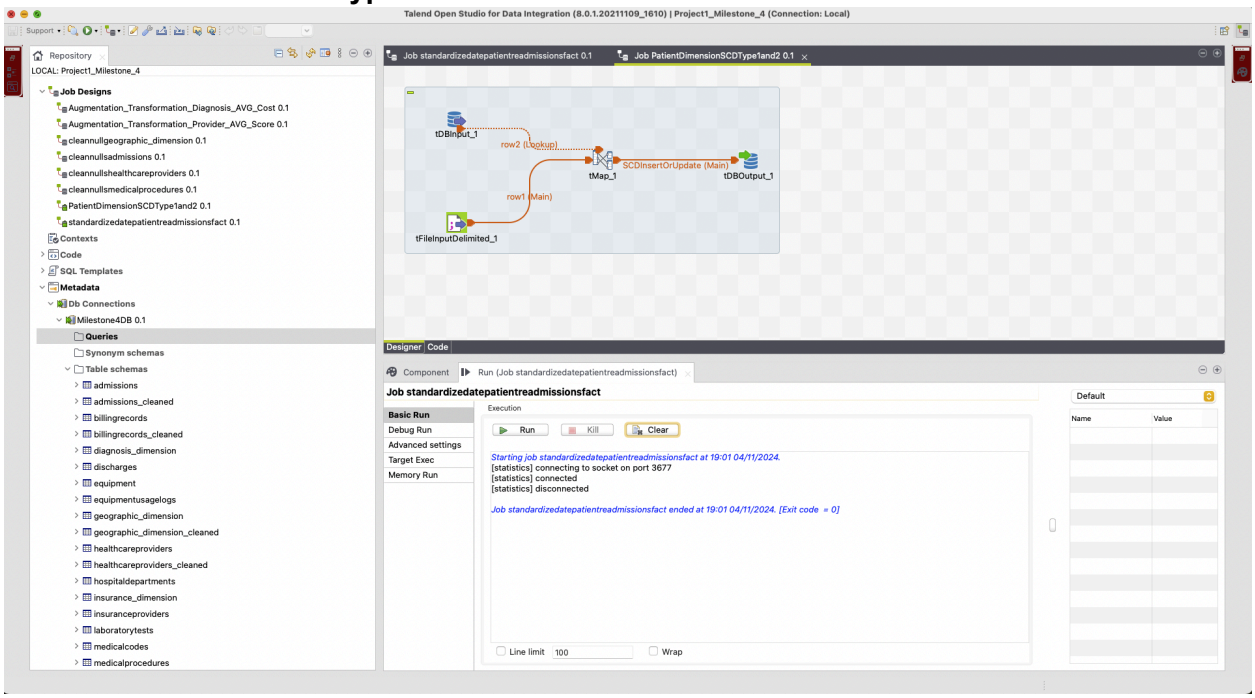



- **Standardized Patient Readmission Fact**

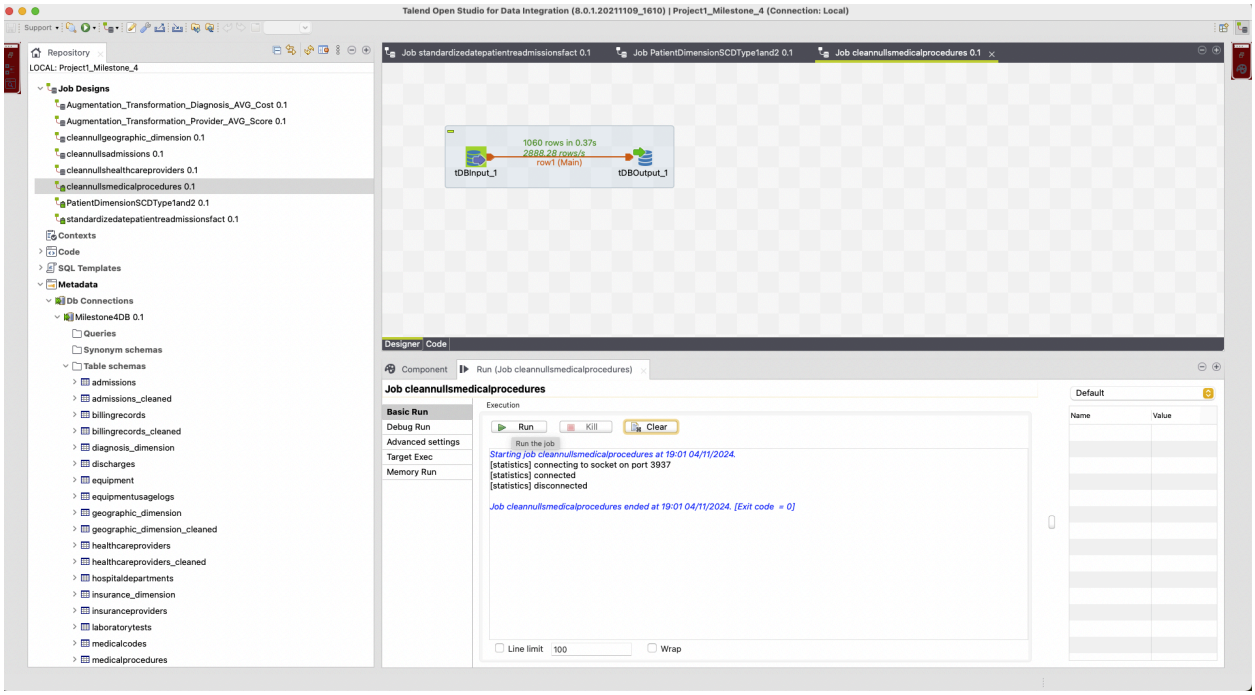


# Final Project 1 Report

- **Patient Dimension SCD Type 1 and 2**

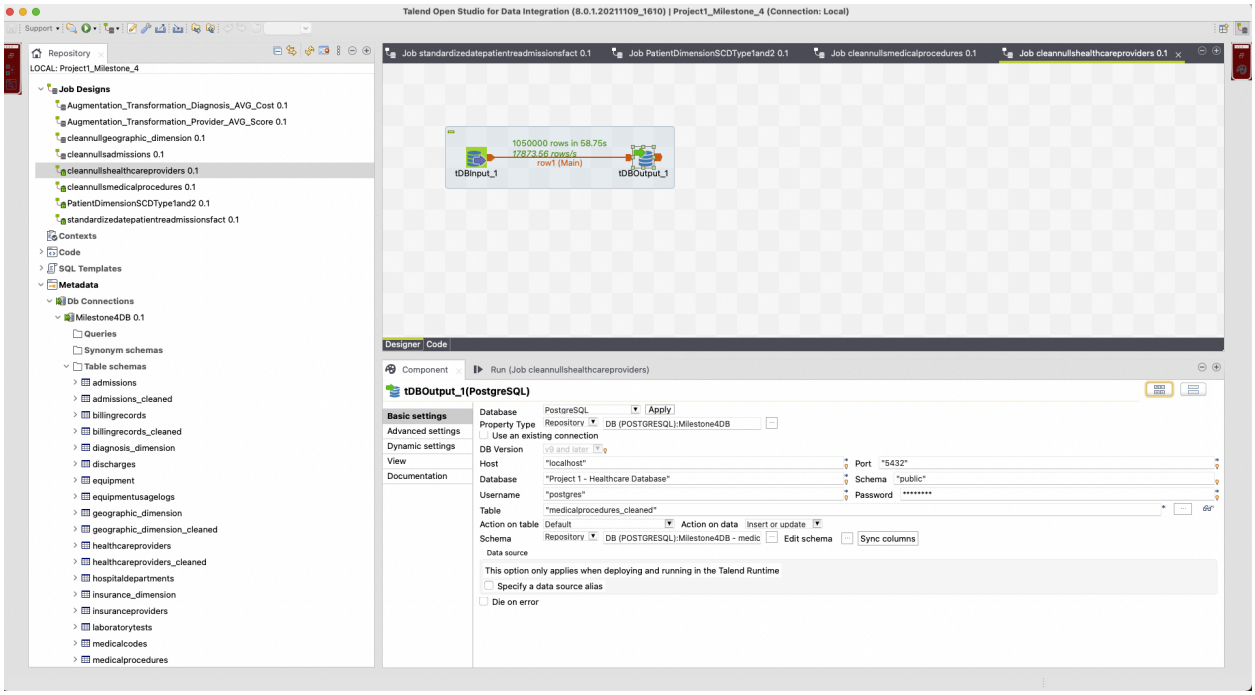



- **Clean Nulls Medical Procedure**

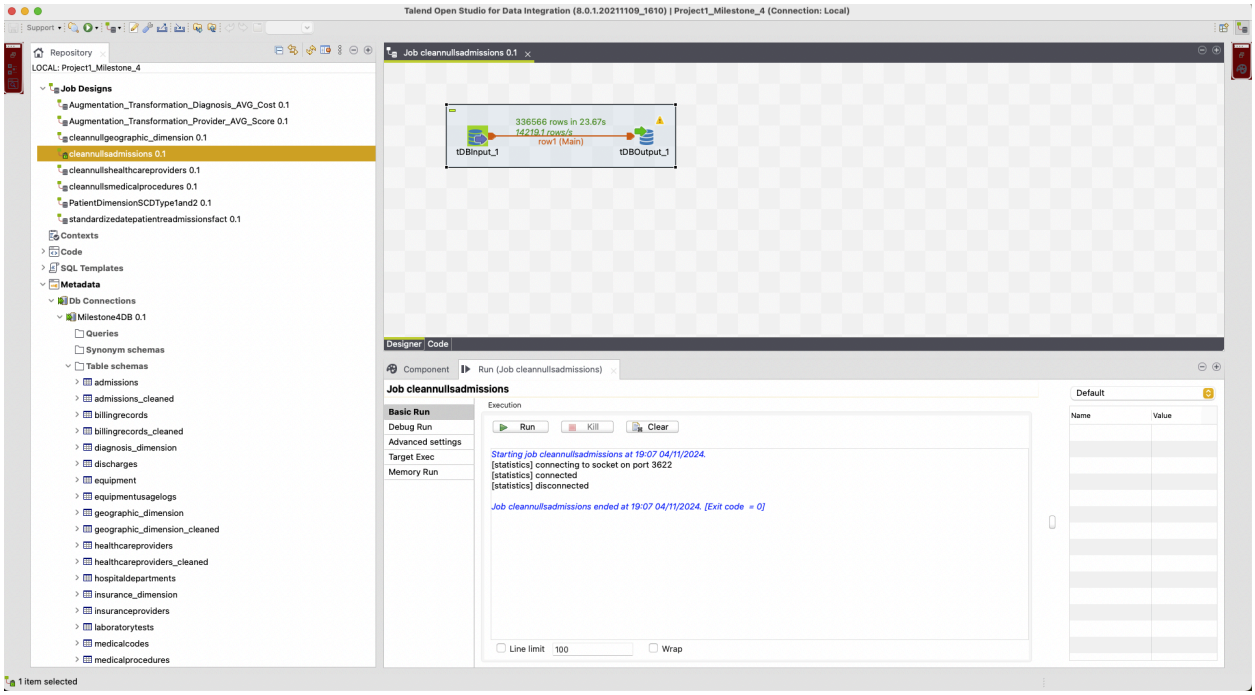


# Final Project 1 Report

- **Clean NULLs in *Healthcare Providers***

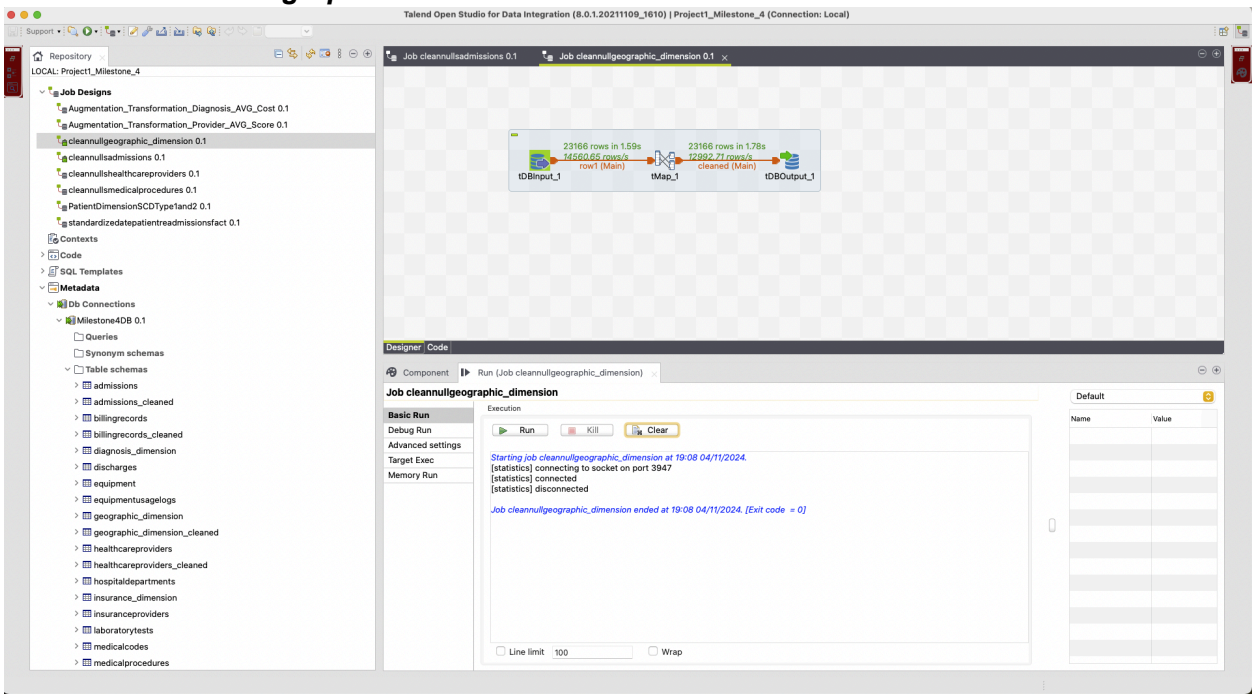



- **Clean NULLs in *Admissions***

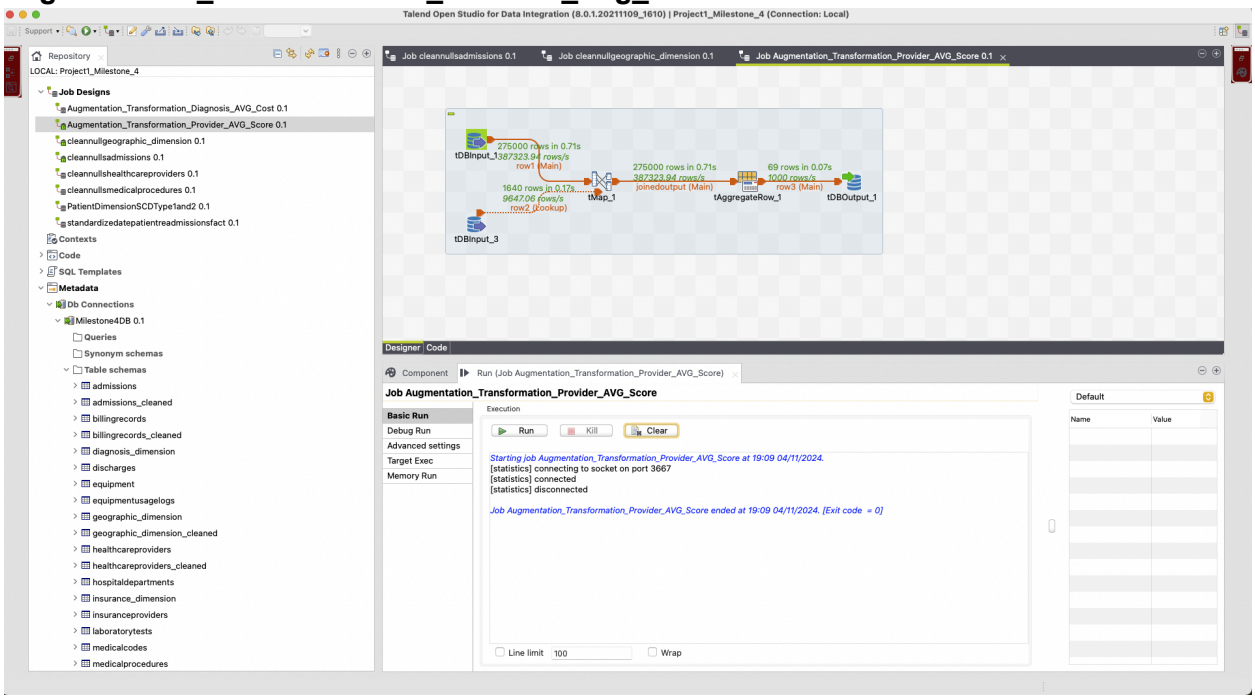


# Final Project 1 Report

- **Clean NULLs in *Geographic dimensions***

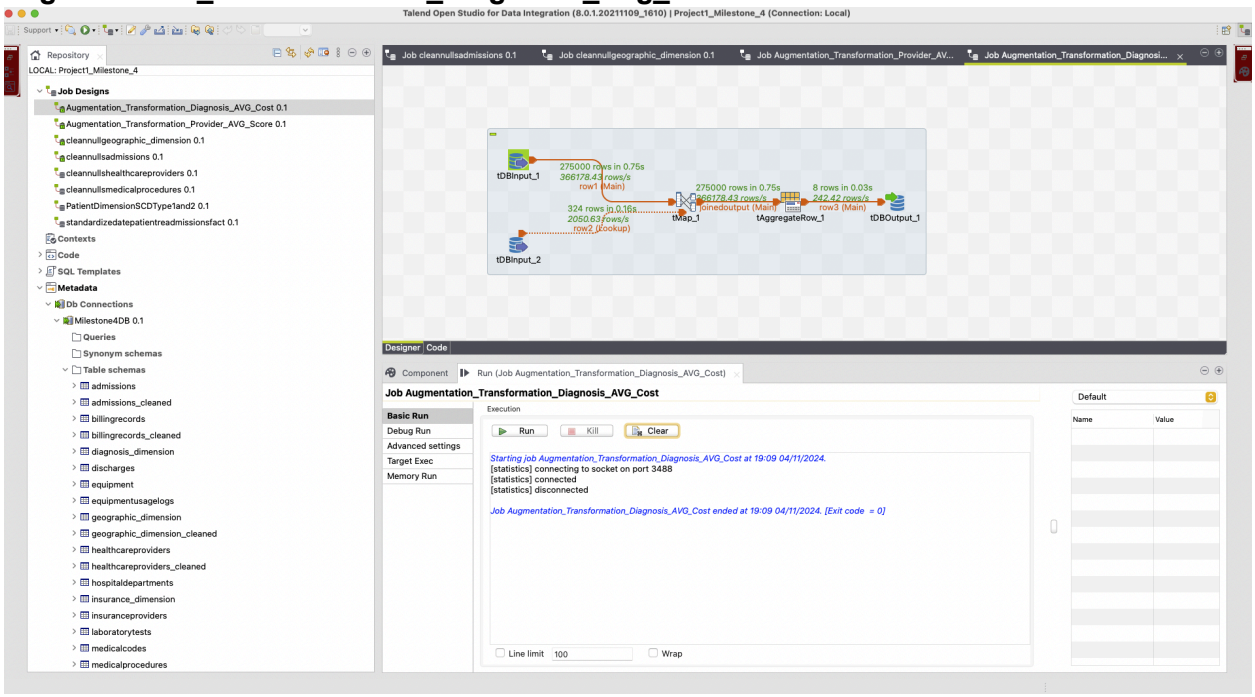



- **Augmentation_Transformation_Provider_Avg_Score**

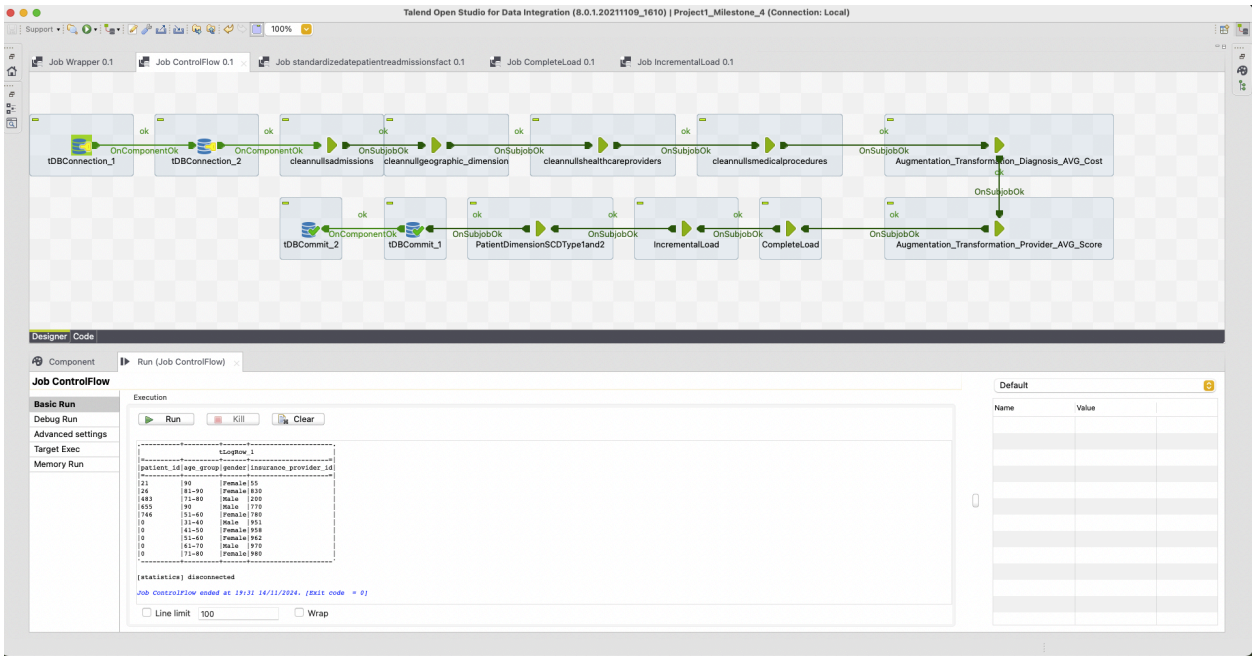


# Final Project 1 Report

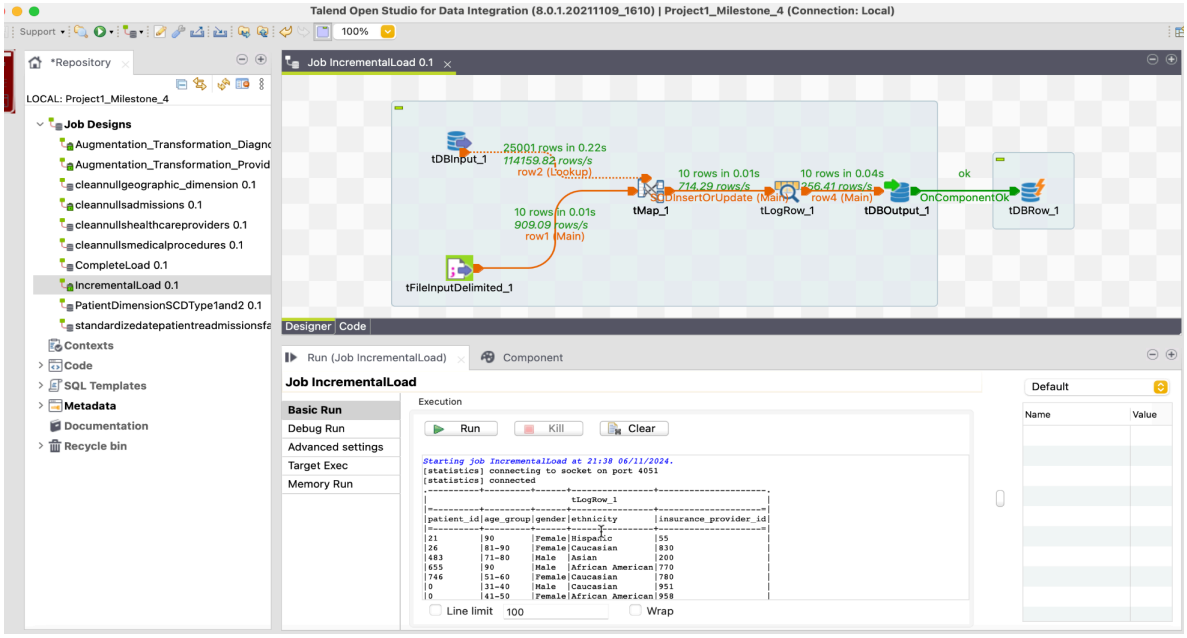
- **Augmentation_Transformation_Diagnosis_Avg_Cost**

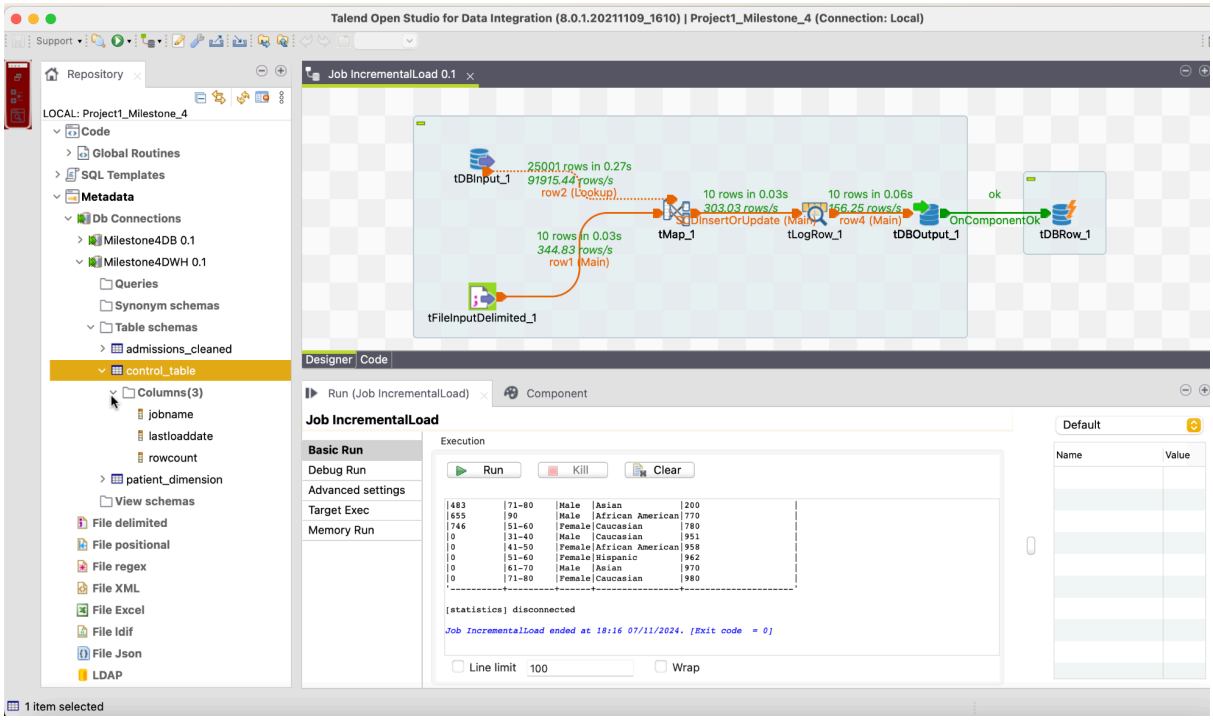



## iii. Loads & Control Flow

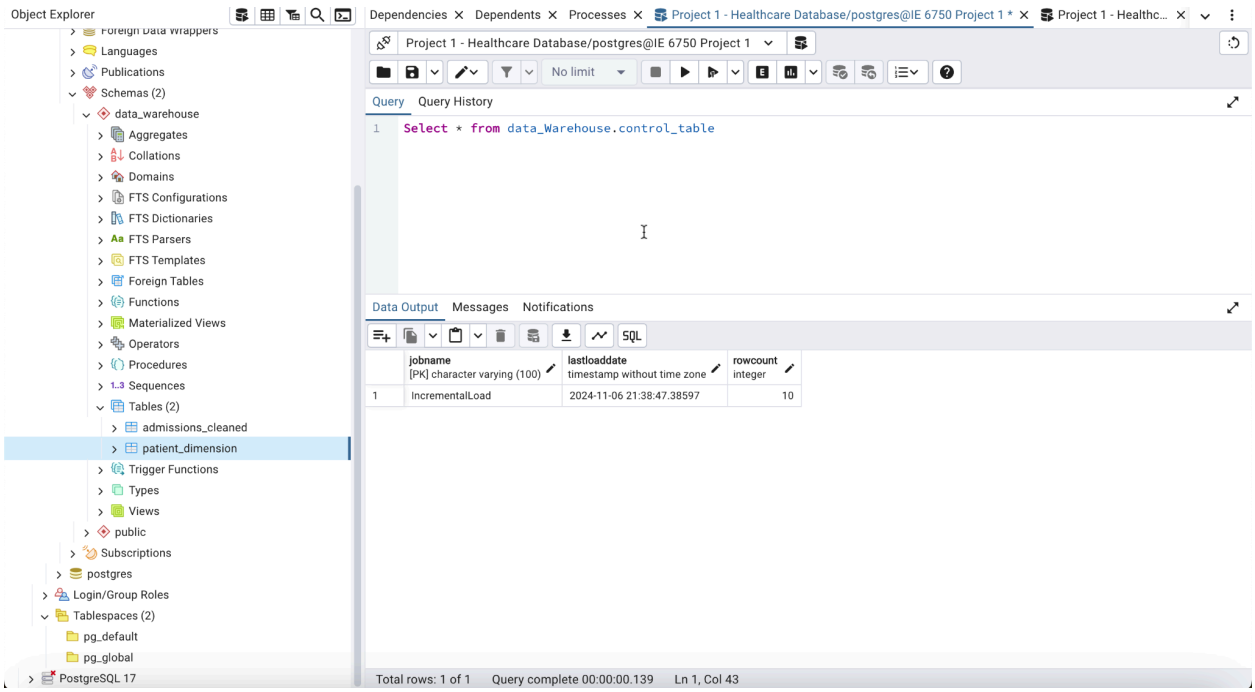


# Final Project 1 Report



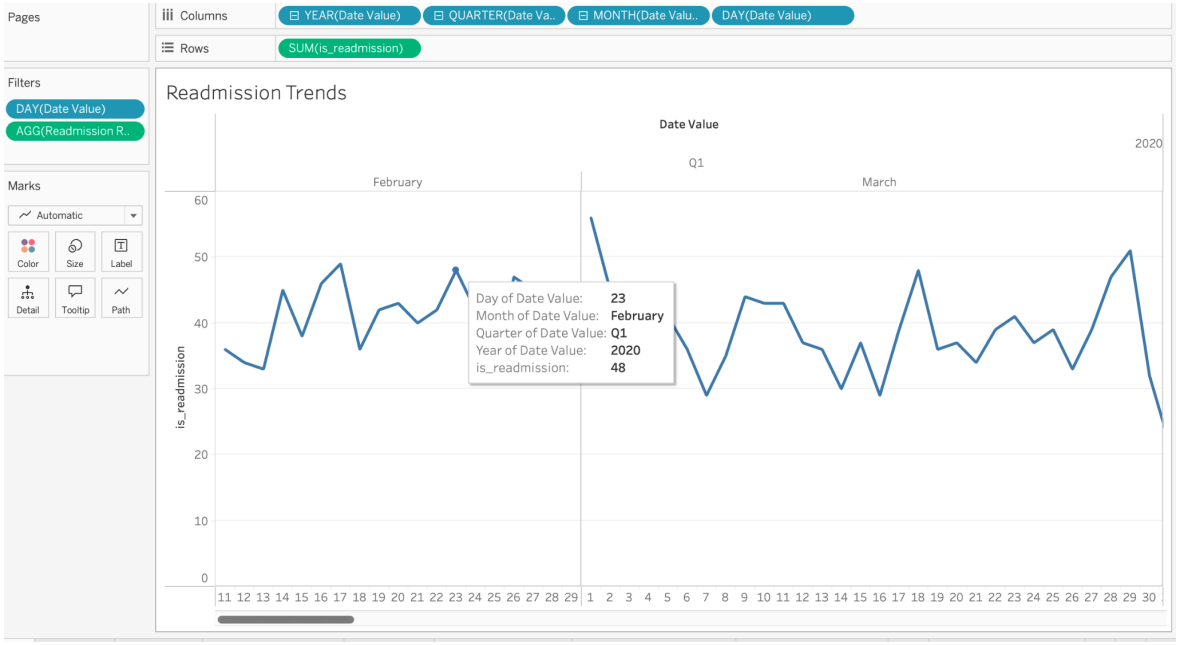



# Final Project 1 Report

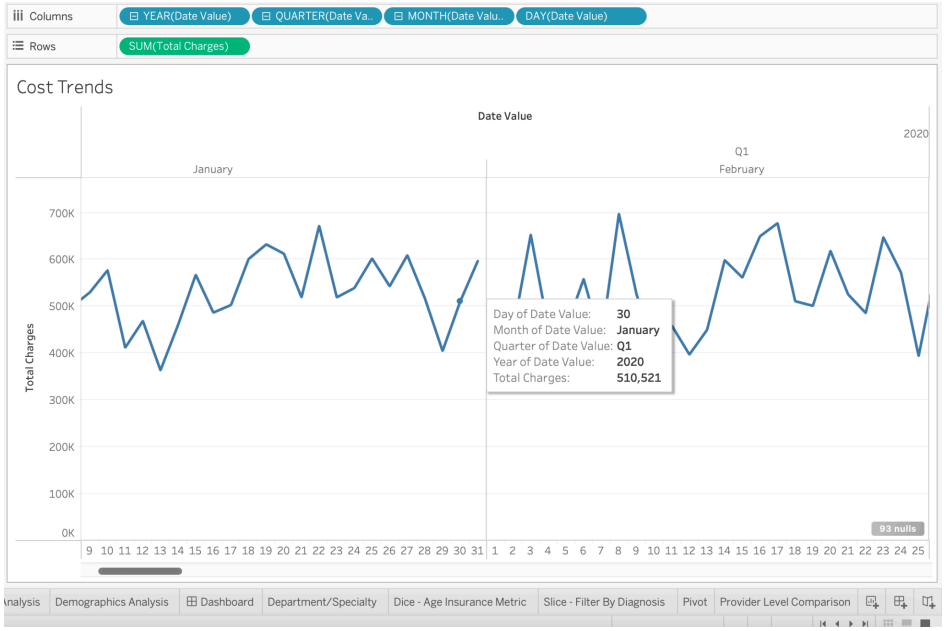


# Final Project 1 Report

**Readmission Trends:** This is one of the KPIs that we used that shows the readmission rates within 30 days. Furthermore, it shows the readmission rates that can be broken down by temporal hierarchy. This shows us

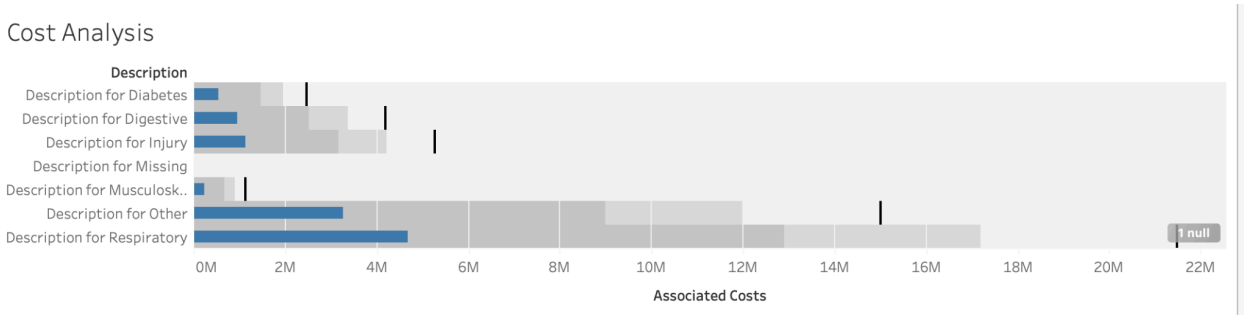



**Cost Trends:** This KPI shows the cost of readmissions over time.

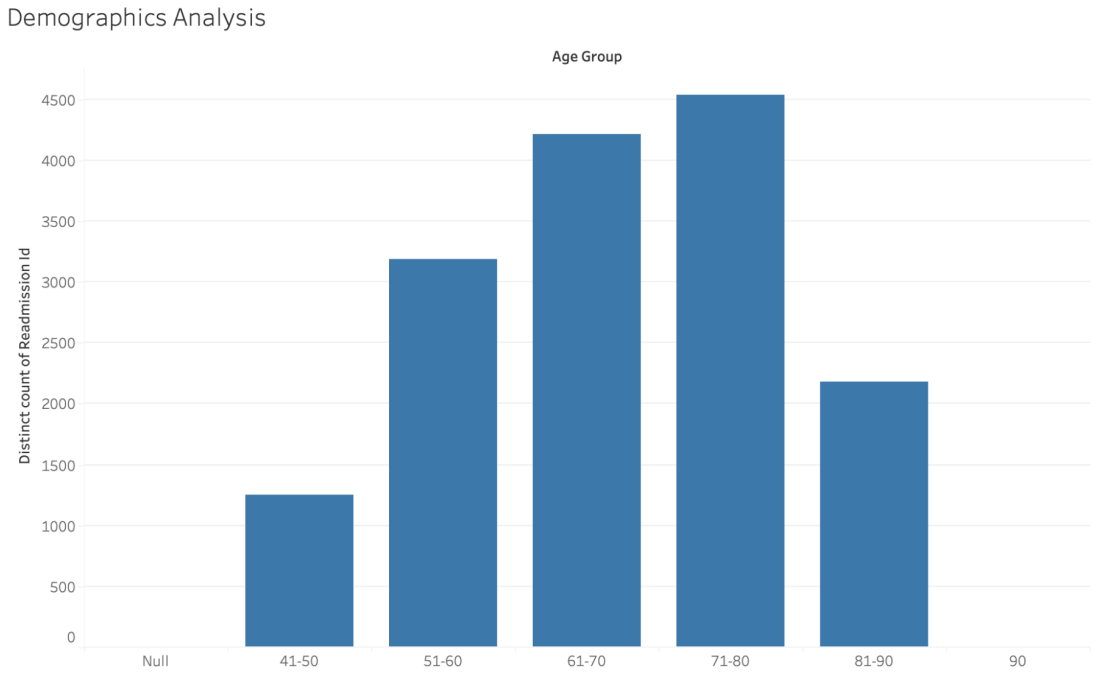



**Cost Analysis:** This KPI shows costs per diagnosis.

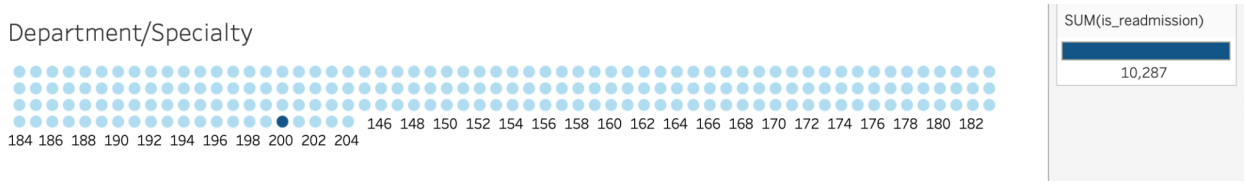
# Final Project 1 Report




**Demographic Analysis:** This demographic analysis arranges the number of readmissions per age group.

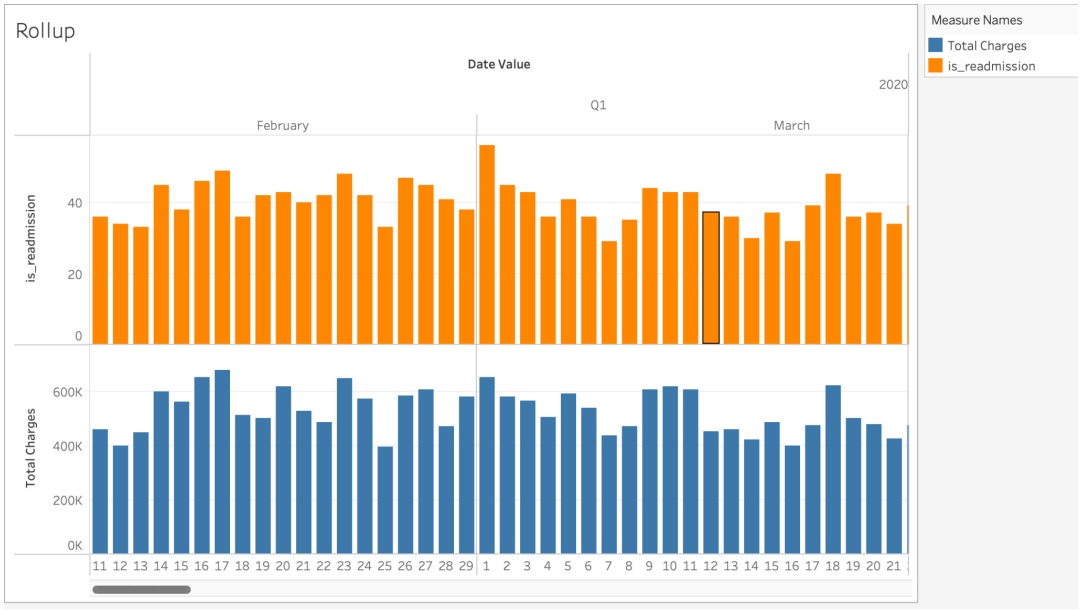



## Drill Down - Department

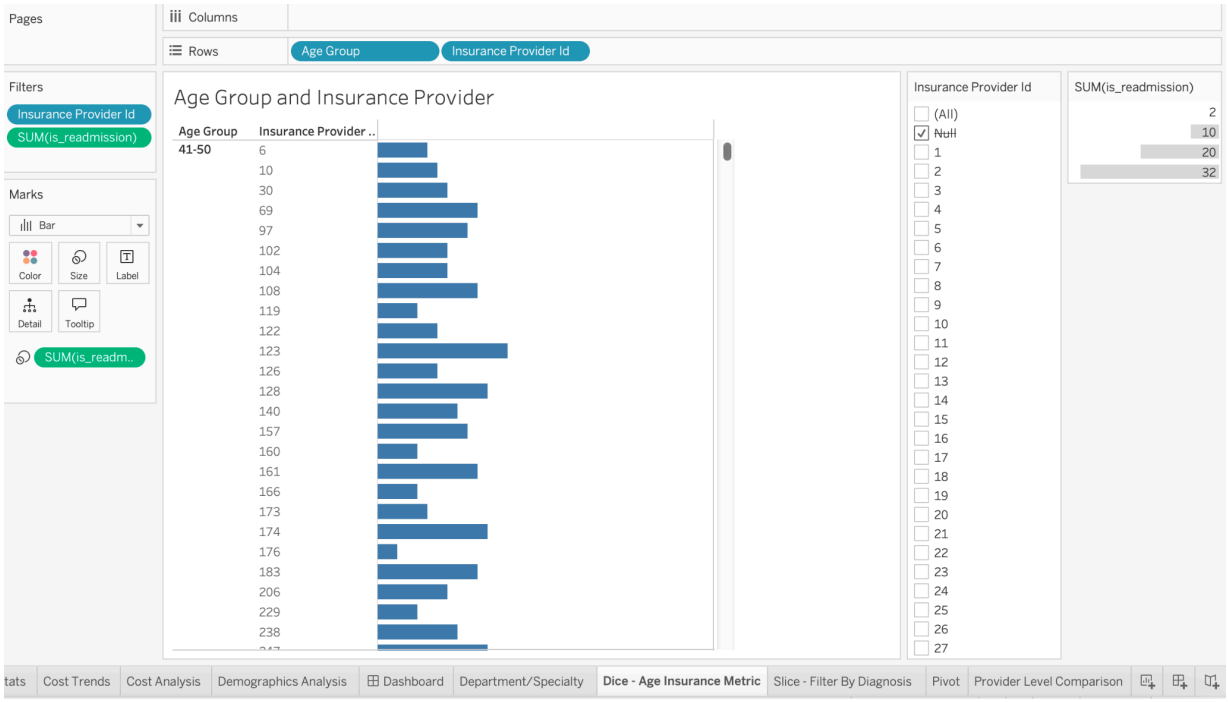


# Final Project 1 Report

**Rollup:** This chart shows the admission rate and charges associated with it per day.

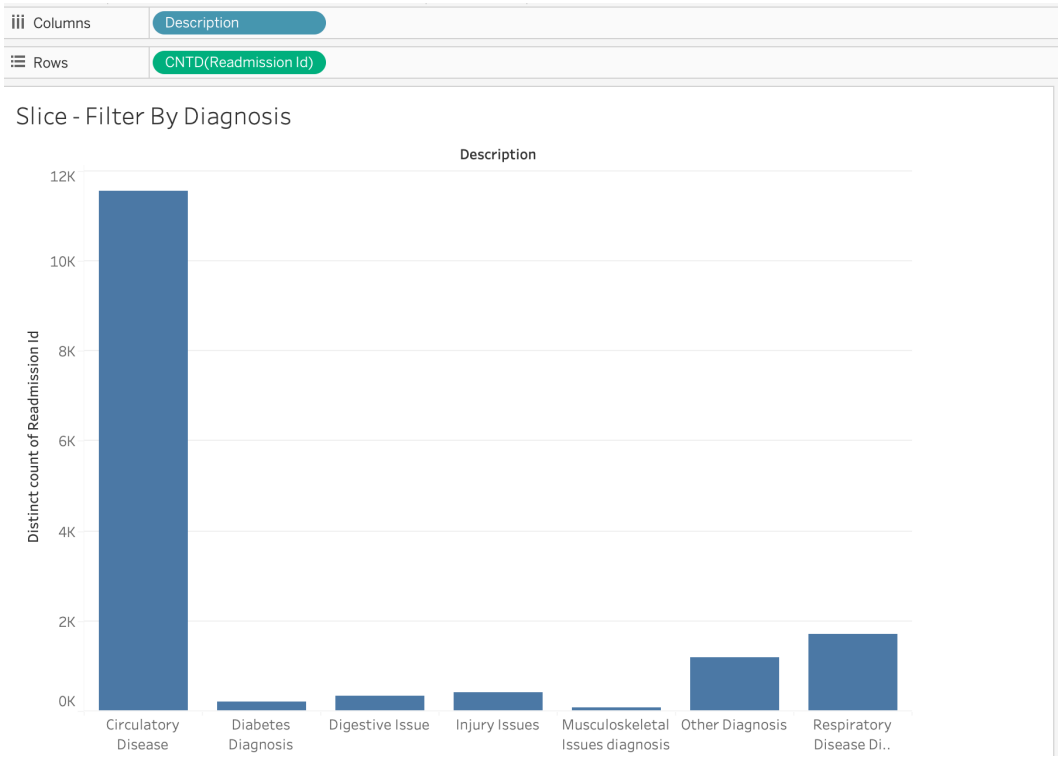



**Age Group and Insurance Provider:** Shows the rate of readmission per age group and insurance provider.




**Slicer:** Filters readmission count by diagnosis type.

# Final Project 1 Report



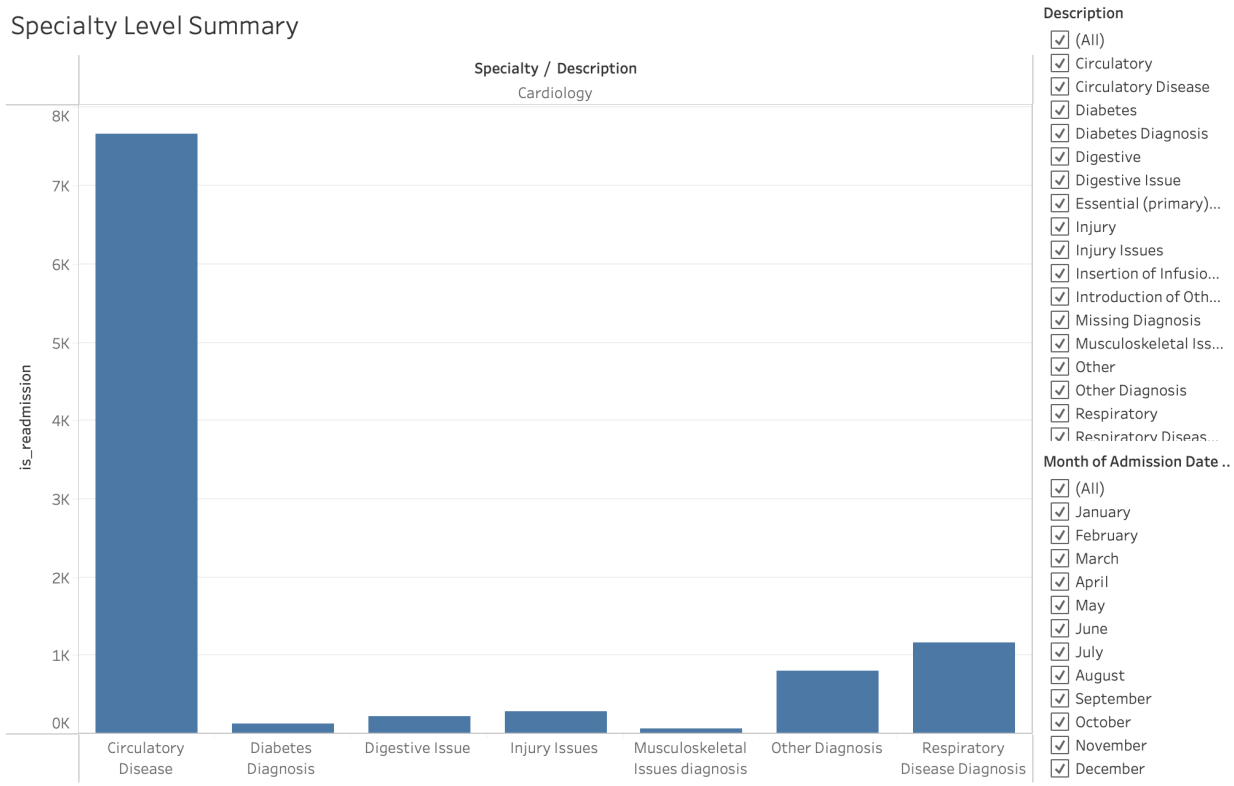

**Pivot:** The following chart shows readmission rate by gender and age to identify demographic trends.

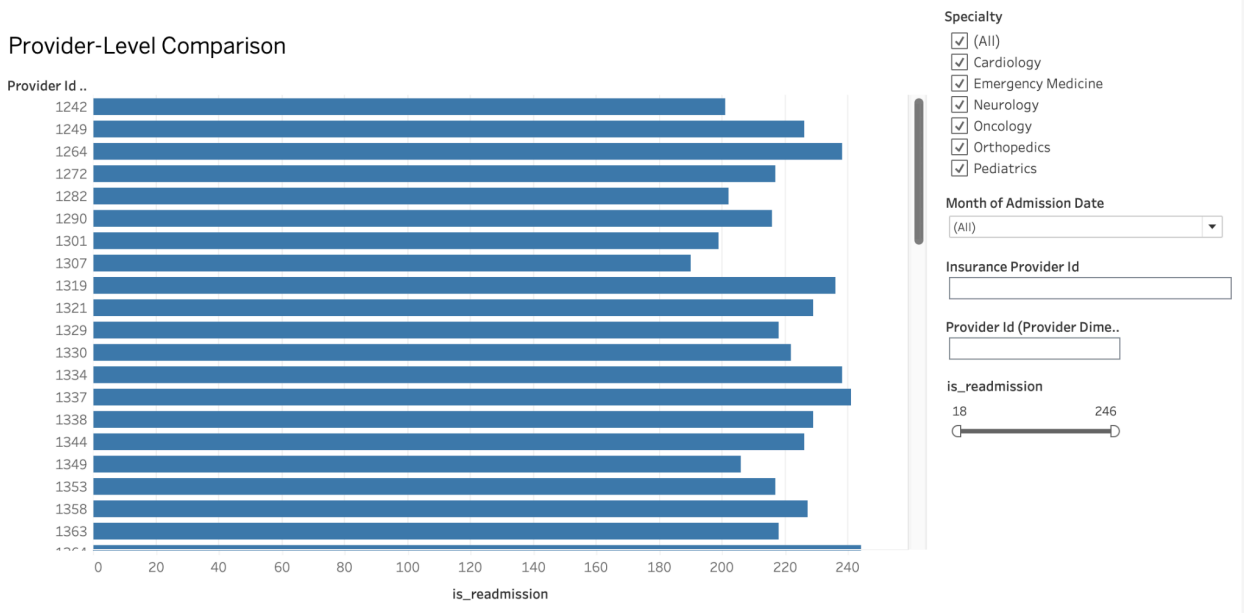
Readmission Rate by Age and Gender

| Age Group | Gender | |
|---|---|---|
| Null | Null | |
| 41-50 | Female | |
| | Male | |
| 51-60 | Female | |
| | Male | |
| 61-70 | Female | |
| | Male | |
| 71-80 | Female | |
| | Male | |
| 81-90 | Female | |
| | Male | |
| 90 | Female | |
| | Male | |
| 90+ | Female | |
| | Male | |

# Final Project 1 Report

**Drill Across:** This chart shows comparison among providers, based on several factors such as their readmission rate, which can be further analyzed month by month. We can also specify a specific number of providers or numbers of readmission. Whereas specialty level summary can drill into readmission rates based on the specialty of the diagnosis. Furthermore, it can do more granular research such as the type of diagnosis and by month.



# Final Project 1 Report

**Insights:**
From these visualizations, some of the key insights we have gained are that, with the number of readmissions, the costs to the hospital also went up. Further, we have noticed that the cardiology department needs an audit, as it was responsible for all of the readmissions. Moreover, we need to provide more care to oldr patients, as the number of readmissions is significantly higher among older patients.

Link to Tableau File and Presentation: MILESTONE 7 ON PREM